A STUDY OF NUMERICAL TECHNIQUES FOR THE INITIAL VALUE PROBLEM OF GENERAL RELATIVITY

by

MATTHEW WILLIAM CHOPTUIK

B.Sc., Brandon University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE in THE FACULTY OF GRADUATE STUDIES Physics Department

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

November 1982

© Matthew William Choptuik, 1982
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Physics

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date November 24, 1982
Abstract

Numerical relativity is concerned with the generation of solutions to Einstein's equations by numerical means. In general, the construction of such a spacetime is accomplished in two stages: 1) the determination of initial data which is specified on a single spacelike hypersurface and satisfies four initial value equations, and 2) the evolution of the initial data to generate the spacetime or some portion of it.

One of the key problems is the development of efficient algorithms for the solutions of these equations, as they are sufficiently complex to tax the fastest present computers. This thesis presents a comparison of various algorithms for the solution of the initial value equations, concentrating on the recently developed multi-grid method.

The specific problem examined has been previously studied by Bowen, Piran and York. Their initial data has been interpreted as representing "snapshots" of three new families of black holes. Three of the four initial value equations possess analytic solutions. The remaining 2-dimensional non-linear partial differential equation is solved numerically in this thesis using finite difference techniques.

The performance of the multi-grid method, with respect to three more well-known methods, is evaluated through numerical experiments. The speed and reliability of the multi-grid algorithm are found to be very good. In addition, the results which had been previously calculated numerically by Piran are
essentially reproduced, with the correction of some errors in that work. Possible extensions of the work to more complex initial value problems are also discussed.
# Table of Contents

Abstract ........................................................................................................... ii
List of Tables .................................................................................................... v
List of Figures .................................................................................................. vi
Acknowledgement ............................................................................................ vii

1. Introduction ................................................................................................. 1
2. The Initial Value Problem for General Relativity ...................................... 7
   2.1 Splitting spacetime into space and time ................................................... 8
   2.2 Intrinsic and extrinsic curvature ............................................................. 16
   2.3 The initial value equations ..................................................................... 24
   2.4 Preservation of the constraints ............................................................... 30
   2.5 York's approach to the initial value problem ......................................... 32
   2.6 Past work on initial value problems ....................................................... 44
3. A Specific Initial Value Problem ................................................................. 47
   3.1.1 Topology of the slices and simplifying assumptions ........................... 48
   3.1.2 Solution of the momentum constraint .............................................. 49
   3.2 The Hamiltonian constraint and inversion techniques ......................... 51
   3.3 Discretization of the Hamiltonian constraint ......................................... 55
4. The Multi-grid Method ................................................................................ 65
   4.1 Basic multi-grid processes .................................................................... 65
   4.2 The full approximation storage scheme ................................................. 75
   4.3 Solution on the coarsest grid .................................................................. 82
   4.4 Treatment of boundary conditions ......................................................... 83
   4.5 Implementation of the FAS algorithm ................................................. 85
   4.6 Work estimates ....................................................................................... 87
   4.7 Adaptive discretization ........................................................................... 90
5. Numerical Experiments and Results ............................................................ 94
   5.1 Comparison of multi-grid method with other methods ....................... 94
   5.2 Testing of the adaptive multi-grid algorithm ....................................... 100
   5.3 Numerical results for boosted black holes ......................................... 108
   5.4 Numerical results for spinning black holes ....................................... 114
6. Conclusions and Possible Future Applications .......................................... 125

Bibliography ................................................................................................. 130

Appendix A - Basic Numerical Techniques for Elliptic PDE's ....................... 134
   A.1 Discretization ....................................................................................... 135
   A.2 Finite differences .................................................................................. 137
   A.3 Methods for solving linear systems ...................................................... 144
      A.3.1 Direct methods ............................................................................. 145
      A.3.2 Iterative methods ........................................................................ 150
   A.4 Methods for solving non-linear systems .......................................... 158

Appendix B - Implementation of Multi-grid Algorithm ............................... 162
   B.1 Grid organization for adaptive discretization .................................... 162
   B.2 Summary of major multi-grid routines ............................................... 165
   B.3 Relaxation routines ............................................................................. 166
   B.4 Control parameters ............................................................................ 167
   B.5 Listing of multi-grid program ......................................................... 169
List of Tables

I. Multi-grid results for modified test problem ........97
II. Point-SOR results for modified test problem ........98
III. Line-SOR results for modified test problem ........98
IV. Newton-ND results for modified test problem ........99
V. Testing of integration routines .....................101
VI. Effect of varying truncation error parameter ........102
VII. Multi-grid test results for varying momentum ......103
VIII. Total energy versus momentum for boosted holes ...110
IX. Various energies of spinning black holes ............124
List of Figures

1. Spacetime Displacement in 3+1 Language .......... 14
2. Extrinsic Curvature Example .................. 21
3. Domain of Boundary Value Problem ............ 58
4. Uniform Grid for Discretization of Boundary Value Problem ............... 61
5. FAS Algorithm .................................. 86
7. Pre-CGC Smoothing of Residuals .............. 106
8. Post-CGC Smoothing of Residuals ............ 107
10. S Dependence of Conformal Factor of Boosted Holes ........ 111
11. R Dependence of Conformal Factor of Boosted Holes ........ 112
12. Angular Dependence of Conformal Factor of Boosted Holes .............. 113
13. Energy vs. Momentum for Spinning Black Holes .... 115
14. S Dependence of Conformal Factor of Spinning Holes 116
15. R Dependence of Conformal Factor of Spinning Holes 117
16. Angular Dependence of Conformal Factor of Spinning Holes .............. 118
17. Conformal Factor of Spinning Black Hole (J=1000) .... 119
18. Angular Momentum and Energy Parameters for Spinning Holes .............. 122
19. Discretization of Domain of Boundary Value Problem 136
20. Extra Grid Points - Discretization of Boundary Conditions .............. 142
21. Natural Ordering of a 10x10 Grid ............ 147
22. Nested Dissection Ordering of a 10x10 Grid .......... 148
23. Non-uniform Discretization and Associated Tree Structure .............. 163
Acknowledgement

I would like to thank Dr W.G. Unruh for suggesting the topic of this thesis and for his substantial contributions to all parts of this work. I would also like to thank Dr U. Ascher for introducing me to the multi-grid method. Finally, I would like to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada.
CHAPTER 1

Introduction

In the six and a half decades since Einstein presented his field equations, general relativity has become widely accepted as the "best" theory of gravitation. However, despite the successes of the theory and the vast amount of effort which has been devoted to its study, relatively little is known about the consequences of general relativity in situations involving a high degree of asymmetry or strong gravitational fields. Situations of this type, such as the collapse of a star to form a black hole, the collision of two black holes, or a supernova explosion are of great interest to astrophysicists and general relativists alike. Due to the complexity and non-linearity of the field equations in such scenarios, current analytic techniques are generally unable to provide acceptable solutions. This has prompted the development of a new branch of general relativity which is concerned with the solution of the field equations by numerical means. Numerical relativity, as the field is called, shows considerable promise in providing information about complex, physically realistic gravitational processes.

The theoretical framework on which most of the current work in numerical relativity relies was developed in the late
1950's and early 1960's by Arnowitt, Deser and Misner and is summarized in their 1962 work "The Dynamics of General Relativity" [1]. The ADM formalism involves a decomposition of spacetime into "space" and "time" so that the dynamical nature of general relativity can be examined. Roughly speaking, in this formalism a 4-dimensional spacetime is regarded as an infinitely extended stack of 3-dimensional spacelike hypersurfaces. The configuration of any particular hypersurface is interpreted as representing the dynamical state of the spacetime at a particular "instant". The complete spacetime is then the "time history" of a single spacelike slice.

When written in the ADM formalism, the field equations separate into two distinct groups. Four of the ten equations involve quantities which may be defined on a single hypersurface and thus represent equations of constraint on the quantities characterizing a spacelike slice. The other six equations govern the "time evolution" of these quantities. The problem of producing a solution to the field equations can therefore be separated into two stages: 1) the determination of initial data which satisfies the constraint, or initial value, equations 2) the evolution of the initial data to produce the complete spacetime or some portion of it.

At the present time, the initial value problem for general relativity is by far the better understood of the above two sub-problems. York and various collaborators have developed a procedure which provides a "covariant" separation of the initial data into freely specifiable and determined-from-
constraint pieces. This process, for a general initial value problem, allows the constraint equations to be written as a set of four coupled non-linear elliptic partial differential equations for four "potentials" which can be interpreted as the relativistic generalization of the single Newtonian gravitational potential. Methods for approximately solving non-linear elliptic systems exist and in principle, there would seem to be no reason why a general initial value problem could not be solved by such means. The situation is not the same for the evolution problem where many questions concerning both the theoretical nature of the problem and practical methods for its solution, remain unanswered.

Although the numerical solution of elliptic equations, such as those which arise in the initial value problem, is completely feasible in principle, there are practical limitations on the size and type of problem which can be satisfactorily solved at present. Even with the rapid increase in the speed and memory capacity of computers, it is still an easy task to pose a numerical problem whose solution is beyond the capability of any existing machine. For this reason, a great deal of research is still being devoted to the search for efficient methods for solving elliptic PDE's numerically. In addition, there are other factors which must be considered here as in the solution of any problem by numerical means. Among these are the reliability of the numerical results and the ability to extend the solution method to more general problems.

In this thesis, a fairly simple initial value problem is
studied. The problem was originally formulated and partially solved by Bowen and York [3]; the solution was completed by Piran and York [41]. The initial data found has been interpreted as representing "snapshots" of three new families of black holes. It is hoped that the techniques used to determine the data will be useful in the solution of an initial value problem for a non-head-on collision of two or more black holes. The problem was re-examined by the author for three main reasons. Most importantly, the problem served to introduce the author to the field of numerical relativity. Secondly, the study was intended to provide a check of the results which previously had been obtained numerically. Finally, the problem served as a "test problem" for an investigation of a numerical technique which may prove useful in the solution of more complicated initial value problems.

The remainder of this introductory chapter is devoted to an outline of the material contained in this thesis.

Chapter 2 begins with a review of the formalism which is used to study general relativity from a dynamical viewpoint. The 3+1 decomposition of a spacetime into a family of spacelike hypersurfaces is described along with the quantities which characterize the decomposition. The concepts of metric, parallel transport and curvature are examined in the 3+1 language, followed by a derivation of the initial value equations. York's approach to the initial value problem is then presented and the chapter concludes with a review of some of the initial value problems which have been previously studied.
In Chapter 3, the specific initial value problem studied for this thesis is discussed. The work previously done on the problem, which includes analytic solutions of three of the four constraint equations, is reviewed. The remaining initial value equation is posed as a boundary value problem. Using finite difference techniques, this boundary value problem is then approximated by a discrete problem suitable for solution by numerical means.

Chapter 4 is concerned with a description of the multigrid method for the numerical solution of discretized boundary value problems. This relatively new technique, developed over the past decade primarily by Brandt and co-workers, has been successfully applied to many problems from various fields of study, but has not been used previously in numerical relativity. The method can be very efficient in comparison to some of the more well-known techniques such as SOR (successive over-relaxation) but requires a considerable amount of effort to implement.

In Chapter 5, the results of some numerical experiments, as well as the results of the numerical solution of the problems described in Chapter 3, are presented. Using test problems of varying size, the performance of the multi-grid method is compared to that of three other methods. The multi-grid algorithm is also evaluated using another test problem having an exact solution. Finally, the numerical results for the three new families of black holes are presented and compared with the results which had been previously calculated.
by Piran.

Chapter 6 contains some conclusions about the study and a discussion of possible future extensions of the work. Following this are two appendices: Appendix A reviews some of the techniques used in the numerical solution of elliptic PDE'S. Finite difference methods are briefly discussed as are some of the methods for solving the systems of algebraic equations which result from the discretization of boundary value problems. Appendix B contains a listing of the multi-grid program used to obtain the results discussed in Chapter 5 as well as some of the details of the implementation.
CHAPTER 2

The Initial Value Problem for General Relativity

The main purpose of this chapter is to review the formalism used to study the initial value problem for general relativity from a space-plus-time (3+1) viewpoint. It is assumed that the reader is familiar with the "basics" of general relativity and differential geometry. This includes the notions of metric, parallel transport and curvature, as well as some familiarity with the field equations. Most of the material in sections 2.1 - 2.3 can also be found in Chapter 21 of Gravitation (Misner, Thorne, and Wheeler [27]), hereafter referred to as MTW. The major reference for sections 2.4 - 2.5 is the paper The Initial Value Problem and Beyond (Piran and York [41]). MTW conventions are used throughout this thesis. Units are chosen such that $G=c=\hbar=1$, and the spacetime metric has signature $(-+++)$. Quantities having the superscript $^{(4)}$ are defined on the spacetime manifold; those denoted by $^{(3)}$ are defined on a spacelike hypersurface of the spacetime. Greek (Latin) indices take on the values $0,1,2,3$ ($1,2,3$); summation conventions for both types of indices are employed. The reader may notice a shift in viewpoint as the chapter progresses. In the first four sections, the existence of some spacetime which satisfies Einstein's equations is assumed. After the
appropriate initial data for this hypothetical spacetime has been identified, and the initial value equations have been presented, attention is focussed on the problem of determining initial data which may be used to construct a generic spacetime.

2.1 Splitting spacetime into space and time

As mentioned in the introductory chapter, the 3+1 approach to general relativity involves a decomposition of the 4-dimensional spacetime manifold into an infinite family of edgeless 3-dimensional spacelike hypersurfaces. (A hypersurface is spacelike if and only if the displacement between an arbitrary pair of distinct points of the surface is spacelike.) There is a great deal of arbitrariness in the way one can slice up a spacetime into spacelike hypersurfaces. Specifically, it is the same freedom one has in choosing a set of coordinate functions $x^M$ on the spacetime manifold such that the tangent vector field to one of the coordinates, $x^o \equiv t$, is everywhere timelike. Having chosen such a coordinate system, the spacelike slices $\Sigma(t)$ are just the $t$=constant surfaces. With $t$ fixed, the remaining three coordinates, $x^i$ serve as a particular internal labelling of $\Sigma(t)$. Actually, a single set of spatial coordinates $x^i$ may not be sufficient to cover a hypersurface completely. This will not affect the following analysis in any way, since as will be shown, a given hypersurface may be characterized by quantities which are independent of the internal labelling. On the other hand, the
The $x^0 \equiv t$ coordinate must be global in the sense that the slices it determines completely fill the spacetime. (See reference [10] for a more detailed discussion.)

Any hypersurface $\Sigma(t)$ chosen in the above manner is assumed to be a well-defined differentiable manifold which may be analyzed using the same techniques of differential geometry that are applied to the spacetime itself. The notions of a point $P$ of the hypersurface and a curve $\lambda(P) : P \in \Sigma(t) \to \mathbb{R}$ are taken as primitive. The local spatial coordinates $x^i$, of the 3+1 coordinate system are three families of such curves. The tangent vector fields $\hat{e}_i$ to the coordinate curves form a spatial basis at each point on the slice which is independent of the way the coordinate system varies away from the hypersurface. Any other vector field, $A^i$ which resides in the slice, may be expressed in terms of its components $A^i$ in the basis $\hat{e}_i$

$$A^i = A^i \hat{e}_i \quad (2.1)$$

In a similar fashion, one can also introduce fields of one-forms on the hypersurface which are written in terms of their components in an arbitrary one-form basis $d\omega^i$. For example

$$B^i = B_i (3) d\omega^i \quad (2.2)$$

The existence of the natural scalar product, $<,>$ between vectors and one-forms of the hypersurface is assumed. If $\hat{e}_i$ is a coordinate basis of tangent vectors then the coordinate one-
form basis, $dx^j$ is dual to $e_i$. That is

$$\left< (3) dx^j, e_i \right> = \delta^j_i \quad (2.3)$$

Then the contravariant components $A^i$ in (2.1) are given by

$$A^i = \left< (3) dx^i, (3) A \right> \quad (2.4)$$

Similarly, the covariant components $B_i$ in (2.2) are

$$B_i = \left< (3) B, e_i \right> \quad (2.5)$$

The fundamental quantity which characterizes the hypersurface, $\Sigma(t)$, is a positive definite 3-metric, $g$, which determines the lengths of tangent vectors and defines natural mappings from vectors to one-forms and vice versa. The covariant components $g_{ij}$ of the 3-metric determine the line element of the slice - the displacement between two points $P$ and $Q$ in $\Sigma(t)$ having spacetime coordinates $(t,x^i)$ and $(t,x^i+dx^i)$ respectively is given by

$$ds^2 = \text{(distance from P to Q)}^2 = \left< (3) g_{ij} dx^i dx^j \right> \quad (2.6)$$

The associated contravariant components, $g^{ij}$, of the 3-metric satisfy

$$g^{ij} g_{jk} = \delta^i_k \quad (2.7)$$
The quantities $g^{ij}$ and $\tilde{g}_{ij}$ may be used to "raise and lower indices" of geometrical objects (tensors) defined on the hypersurface.

Because the hypersurface $\Sigma(t)$ is embedded in a higher dimensional manifold, it is instructive to examine the relationship between objects defined on the surface and those which reside in spacetime itself. The quantities of greatest interest are those which may used to characterize the hypersurface itself or which describe how the slice is actually embedded in the spacetime. Moreover, to keep things as general as possible, such quantities should not depend on how the hypersurfaces are labelled internally - that is, they should transform as tensors under any coordinate change that leaves the $t=$constant surfaces unchanged (change of spatial basis). First, consider an arbitrary 4-vector field

$$^{(4)}\bar{A} = ^{(4)}A^\mu \tilde{e}_\mu$$

$$= ^{(4)}A^i \tilde{e}_i + ^{(4)}A^0 \tilde{e}_0$$

(2.8)

Clearly, this object can not be considered to be defined on a single hypersurface because of the presence of the component $^{(4)}A^0 \tilde{e}_0$. An associated 3-vector field $^{(3)}\bar{A}$ residing on a slice could conceivably be constructed from $^{(4)}\bar{A}$ by "killing" the unwanted component, making the identification

$$^{(3)}\bar{A}^i = ^{(4)}A^i$$

However, the components $^{(4)}A^i$ are determined in terms of scalar products with the spacetime basis of one forms $^{(4)}dx^\mu$
\[(4)^A_i = (4) \langle (4) \, dx^i, (4) \bar{A} \rangle \quad (2.9)\]

and this basis depends on the way in which the spatial coordinate system varies away from the hypersurface. Thus the components \((4)^A_i\) do not form the components of an object which transforms as a 3-vector under change of spatial coordinates. However, the spatial components \((4)^A_i\) of a spacetime one-form field \((4)^A\)

\[(4)^A_i = (4) \langle (4) \bar{A}, \bar{e}_i \rangle \quad (2.10)\]

may be identified with the three components of a one-form defined on the hypersurface since, as stated previously, the tangent vectors \(\bar{e}_i\) do not depend on the way the coordinate system varies away from the slice. Thus, under a change of spatial coordinates, the components \((3)^A_i = (4)^A_i\) will transform as desired. This statement may be extended to the general case of a purely covariant tensor field \((4)^T_{\mu\nu\sigma}\) on the spacetime manifold - the spatial components \((4)^T_{ij\kappa}\) of such a tensor at any point \(P\) having coordinates \((t,x^i)\) form the components of a tensor \((3)^T_{ij\kappa}\) which is intrinsic to the hypersurface \(\Sigma(t)\). In particular, the spatial components \((4)^{gi}_{ij}\) of the spacetime metric may be identified with the components \((3)^{gi}_{ij}\) as will now be shown.

To determine the full relationship between the components \((4)^{g}_{\mu\nu}\) of the spacetime metric and those of a 3-metric \((3)^{g}_{ij}\) in a 3+1 coordinate system, consider two points \(P\) and \(Q\) having coordinates \(x^\mu\) and \(x^\mu + dx^\mu\) respectively. These points then
reside on distinct but "nearby" hypersurfaces. The spacetime displacement, or proper time, between the two points is given by

\[ (A) \Delta s^2 = (A) g_{\mu\nu} \, dx^\mu \, dx^\nu \]  \hspace{1cm} (2.11)

There is another way of calculating this distance as is suggested by Figure 1. On the surface \( \Sigma(t) \) locate the point \( P' \) which lies directly "beneath" \( Q \) in the sense that a small displacement from \( P' \) in the direction normal to the hypersurface \( \Sigma(t) \) takes one to the point \( Q \). Define the function \( N(t,x^i) \) such that \( N(t,x^i) dt \) is the proper distance between \( P' \) and \( Q \). In general the coordinates of \( P' \) are not \( (t,x^i + dx^i) \). One must introduce three more functions \( N_i(t,x^i) \) such that \( (t,x^i + dx^i + N_i dt) \) are the coordinates of \( P' \). Then, using a 4-dimensional generalization of the Pythagorean theorem, the proper distance from \( P \) to \( Q \) is

\[ (4) \Delta s^2 = (4) g_{ij} (dx_i + N_i dt) (dx_j + N_j dt) - (N_{i} dt)^2 \]  \hspace{1cm} (2.12)

The functions \( N \) and \( N_i \) are called the \textit{lapse} and \textit{shift} functions respectively since \( N \) determines the lapse in proper time between successively labelled hypersurfaces, while \( N_i \) describes the shift in the spatial coordinates between the two slices. Together the four functions are simply a manifestation of the fact that one has chosen a particular coordinate system to slice the spacetime and label the resulting hypersurfaces. If a different slicing or labelling is chosen, then \( N_i \) and \( N \)
Figure 1

Spacetime Displacement in 3+1 Language
must adjust so as to keep (2.12) valid.

The 3 functions \( N^i(t,x^i) \) may be regarded as the components of a 3-vector field residing in the hypersurface \( \Sigma(t) \). The associated covariant components are given by

\[
N_j = \langle^3 q_{ij} \rangle N^i
\]  

(2.13)

Using this relation in (2.12) and comparing the result with (2.11) leads to the identification

\[
\begin{align*}
(4) q_{00} &= (N_k N^k - N^t) \\
(4) q_{0i} &= (4) q_{i0} = N_i \\
(4) q_{jk} &= \langle^3 q_{jk} \rangle
\end{align*}
\]  

(2.14)

This is the ADM expression for the spacetime metric in terms of the 3-metric of a spacelike hypersurface and the lapse and shift functions. The components \((4) g^{\mu\nu}\) of the reciprocal 4-metric may be expressed in this language via the relationship

\[
(4) g^{\mu\nu} (4) g_{\nu\rho} = \delta^{\mu}_{\rho}
\]  

(2.15)

with the result

\[
\begin{align*}
(4) g^{00} &= -\frac{1}{N^t} \\
(4) g^{0i} &= (4) g_{0i} = \frac{N^i}{N^t} \\
(4) g^{jk} &= \langle^3 g^{jk} \rangle = \frac{N^j N^k}{N^t}
\end{align*}
\]  

(2.16)

Another quantity which is useful for describing the embedding of a spacelike hypersurface in a spacetime is the
unit timelike vector field $\mathbf{n}$ which is normal to the hypersurface

\[ (\delta) \quad g^{\alpha \beta} n^\alpha n^\beta = -1 \]
\[ n^\alpha A_\alpha = 0 \]  \hspace{1cm} (2.17)

where $A_\alpha$ are the components of any one-form which can be considered to reside in the hypersurface, that is with $A_0 = 0$. The components $n^\alpha$ may be easily determined by observing that the components $n_\alpha$ of the associated one-form $\mathbf{n}$, $(\langle \mathbf{n}, \mathbf{n} \rangle = -1)$ in the basis $dx^\mu = (dt, dx^i)$ are just

\[ n_0 = N \]
\[ n_i = 0 \]  \hspace{1cm} (2.18)

Then, using (2.16)

\[ n^0 = \frac{1}{N} \]
\[ n^i = -\frac{N^i}{N} \]  \hspace{1cm} (2.19)

A particular coordinate system which is very useful for the derivation of the initial value equations results from choosing the unit normal as the timelike basis vector, which then implies that the lapse function is equal to unity everywhere on the hypersurface.

2.2 Intrinsic and extrinsic curvature

To complete the description of the relationship between a hypersurface and the spacetime in which it is embedded, the concepts of parallel transport and curvature must be examined
in the 3+1 formalism. There are actually three interrelated notions of curvature to be dealt with: 1) the curvature of the spacetime itself, characterized by the spacetime Riemann tensor \( R^\alpha_{\beta\gamma\delta} \), 2) the curvature intrinsic to the hypersurface, described by the 3-dimensional curvature tensor \( R^i_{ij\xi} \), and 3) the curvature which describes how the hypersurface is embedded in the spacetime.

Recall that the curvature at a particular event of spacetime may be determined by examining the change in direction of an arbitrary vector which is parallel transported around a closed loop in a neighborhood of the event. The existence of a well-defined operation of parallel transport is equivalent to the existence of a spacetime covariant derivative operator \( \nabla_\mu \) which is compatible with the 4-metric

\[
\nabla_\mu q^{\alpha\beta} = q^{\alpha\beta}; \mu = 0 = q^{\alpha\beta}; \mu \quad (2.20)
\]

The fact that a vector rotates when carried along a small closed loop is reflected in the non-commutativity of covariant differentiation. Thus, for an arbitrary vector \( ^4\mathbf{W} = \mathbf{W}^\alpha \mathbf{e}_\alpha \), having an associated one-form \( ^4\mathbf{W} = \mathbf{W}_\alpha dx^\alpha \), the components of the Riemann tensor satisfy the following relation

\[
^4 R^\alpha_{\beta\gamma\delta} \mathbf{W}_\alpha = \nabla_\delta \nabla_\gamma \mathbf{W}_\beta - \nabla_\gamma \nabla_\delta \mathbf{W}_\beta \quad (2.21)
\]

The curvature intrinsic to a spacelike hypersurface \( \Sigma(t) \) may be analyzed in the same way, by introducing concepts of
parallel transport and covariant derivative defined with respect to the 3-geometry of the slice. The covariant derivative along the basis vector \( \mathbf{e}_i \), taken with respect to the geometry of the hypersurface is denoted by \( D_i \). It is compatible with the 3-metric

\[
D_i (\overline{\gamma}^j_{\kappa}) = D_i (\overline{\gamma}^j_{\kappa}) = 0 \tag{2.22}
\]

The result of applying this operator to an arbitrary vector \( \overline{\mathbf{A}} = A^j \mathbf{e}_j \) tangent to the hypersurface is

\[
D_i (A^j \overline{\mathbf{e}}_j) = A^j,i \overline{\mathbf{e}}_j + A^j \overline{\gamma}^j_{\kappa} \overline{\mathbf{e}}_\kappa \tag{2.23}
\]

where \( \overline{\gamma}^j_{\kappa} \) are the connection coefficients of the 3-geometry which may be expressed, in any coordinate system, in terms of the 3-metric and its first derivatives

\[
\overline{\gamma}^j_{\kappa} = \overline{\gamma}^k_{ij} \tag{2.24}
\]

Note, that because \( \overline{\gamma}^j_{ij} = \overline{\gamma}^j_{ij} \)

\[
\overline{\gamma}^j_{ij} = \overline{\gamma}^j_{ij} \tag{2.25}
\]

Similarly, for a one form \( \overline{\mathbf{B}} = B_j \mathbf{d}x^i \)

\[
D_i (B^j \mathbf{d}x^i) = B^j,i \mathbf{d}x^i + B^j \overline{\gamma}^j_{\kappa} \mathbf{d}x^\kappa \tag{2.26}
\]
Now consider the application of $\nabla_i$ to the vector $\vec{A} = A^i \vec{e}_i = \vec{A}$

$$\nabla_i (A^i \vec{e}_j) = A^i \nabla_i \vec{e}_j + A_j^{(4)} \nabla_{j_i} \vec{e}_\mu$$  \hspace{1cm} (2.27)$$

The result contains a component,

$$(^4) \langle \sigma, A_j^{(4)} \nabla_{j_i} \vec{e}_\sigma \rangle$$

which is normal to the hypersurface. If this component is "projected out", then what remains is precisely the covariant derivative of $\vec{A}$ taken with respect to the hypersurface. That is

$$D_i (A^i \vec{e}_j) = \nabla_i (A^i \vec{e}_j) - A_j^{(4)} \nabla_{j_i} \langle \sigma \vec{e}_\sigma \rangle \vec{A}$$  \hspace{1cm} (2.28)$$

Having defined a notion of covariant differentiation which is intrinsic to the hypersurface, the components of the 3-dimensional Riemann curvature tensor $^{(3)}R^i_{jk\ell}$ may be determined from the following

$$^{(3)}R^i_{jk\ell} V_i = D_\kappa D_\ell V_j - D_\ell D_\kappa V_j$$  \hspace{1cm} (2.29)$$

where $V^i$ are the components of an arbitrary vector tangent to the slice.

In addition to the intrinsic curvature $^{(3)}R^i_{jk\ell}$ of a hypersurface $\Sigma(t)$, another measure of curvature may be defined on the hypersurface which completely describes the way in which $\Sigma(t)$ is embedded in the spacetime. A simple example
best illustrates the features of this curvature. Consider two sheets of paper, one laying flat, and the other rolled up into a cylinder (Figure 2a). Both sheets have the same intrinsic Euclidean geometry as could be determined by hypothetical observers confined to the sheets, by the lack of rotation of tangent vectors parallel transported, with respect to their 2-geometries, around closed loops (Figure 2b). A 3-dimensional observer, however, sees a definite distinction between the two cases, claiming that the cylinder is "curved" by virtue of the manner in which it is embedded in the higher dimensional space. This extrinsic curvature (or lack of it) may be measured by examining a vector field constructed normally to the 2-dimensional surface. The unit normal vector at the point at which the extrinsic curvature is to be measured is parallel transported with respect to the embedding space a short distance along the surface. The parallel-transported normal is then compared to the normal which already resides at the new position. In the appropriate limits, the difference between these two vectors is another vector tangent to the surface which provides a measure of part of the extrinsic curvature (Figure 2c). To complete the measurement in 2-dimensions would require that the process be repeated by transporting the vector in a different direction. In general, the extrinsic curvature describing the embedding of a surface in a "one-higher" dimensional space is a second rank tensor having no components in the direction of the normal to the surface and can thus be considered to be defined on the surface.

Returning to the case of a spacelike hypersurface
Figure 2
Extrinsic Curvature Example
embedded in spacetime, the covariant derivative of the one-form \( \eta \) has components

\[
\eta_{\alpha \beta} = \eta_{\alpha \beta} - \eta_{\mu} \Gamma^{\mu}_{\alpha \beta}
\]  

(2.30)

The spatial components of the above expression define the covariant components \( K_{ij} \) of the extrinsic curvature tensor defined on the hypersurface

\[
K_{ij} = - (\nabla_{i} \eta_{j} - \eta_{\mu} \Gamma^{\mu}_{\alpha \beta} \eta_{i j})
\]  

(2.31)

(the sign above has been chosen by convention and has no physical significance). Using equation (2.18), this becomes

\[
K_{ij} = - \eta^{(4)} \Gamma^{\alpha}_{ij}
\]  

(2.32)

It follows from the symmetry of the \( \eta^{(4)} \Gamma^{\alpha}_{ij} \) in any coordinate system that

\[
K_{ij} = K_{ji}
\]  

(2.33)

Equation (2.28) may now be rewritten with the help of (2.32) to display the relation between the covariant derivatives of the spacetime and hypersurface, and the extrinsic curvature

\[
\nabla_{i} (A^{j} \xi_{j}) = \mathcal{D}_{i} (A^{j} \xi_{j}) - K_{ij} A^{j} \xi
\]  

(2.34)

For the special case when the arbitrary vector \( \tilde{A} \) above is a
spatial basis vector, this becomes

$$\nabla_i \tilde{\epsilon}_j = D_i \tilde{\epsilon}_j - K_{i;k} \delta^k_j \tilde{n}$$

(2.35)

A closer examination of equation (2.32) reveals an interesting relationship between the extrinsic curvature and the 3-metric

$$K_{ij} = - N \left[ \epsilon^{(4)} g^{\alpha \beta} \gamma_{\alpha \beta}^{\gamma} - \epsilon^{(4)} g^{\alpha \beta} \Gamma_{\alpha \beta}^{\gamma} \right]$$

$$= \frac{1}{2N} \left[ \epsilon^{(4)} g_{ij} + (\epsilon^{(4)} g_{ij})_{;i} - (\epsilon^{(4)} g_{ij})_{;j} - 2N \epsilon^{(4)} \Gamma_{ij} \right]$$

(2.36)

In the special case where the 3+1 coordinate system is chosen such that $N=1$ and $N'=0$ (Gaussian normal coordinates) the above becomes

$$K_{ij} = - \frac{1}{2} \frac{\partial^{(4)} g_{ij}}{\partial t}$$

(2.37)

Thus the extrinsic curvature may be visualized as a "velocity" of the 3-metric in the direction normal to the hypersurface. It must be emphasized, however, that the extrinsic curvature is defined on a single hypersurface as opposed to a "true" velocity of the 3-metric whose determination involves, in principle, specifying the 3-metrics on two nearby slices and then calculating the rate of change via some sort of limiting procedure in which the separate slices approach each other. The fact that two slices are involved implies that this procedure will necessarily involve the coordinate choice which induces the slicing, which in turn implies that the resulting
velocity which is to characterize a single hypersurface will not necessarily be a tensor on the slice.

From a dynamical point of view, the 3-metric components \(^{(3)}g_{ij}\) determine the configuration of a hypersurface and may be interpreted as generalized coordinates which specify the "position" of the hypersurface in the space of all possible positive definite 3-dimensional manifolds. Loosely speaking, the extrinsic curvature components \(K^i_j\) may then be thought of as the momenta dynamically conjugate to the \(^{(3)}g_{ij}\). This interpretation is consistent with simpler dynamical theories such as Newtonian mechanics where the conjugate momenta, although closely related to the time derivatives of the generalized coordinates, do not demand for their specification the notion of giving the state of the system at two infinitesimally separated times. A more complete description of this viewpoint is found in the 1962 ADM paper [1]. The set \(^{(3)}g_{ij}, K^i_j\) completely characterizes the embedding of a hypersurface in a spacetime, or "the state of a spacetime at some instant" and thereby constitutes initial data for the spacetime.

2.3 The initial value equations

Having decided that the 3-metric and extrinsic curvature are appropriate quantities for the complete description of the dynamical state of a spacetime, attention is now focussed on the question of the independence of this initial data. As this section will show, four components of the Einstein tensor,
namely $G^\nu_\mu$, may be written in a $3+1$ coordinate system entirely in terms of the quantities $(_G^\nu_\mu, {{G^i_j}_k})$. Therefore the four field equations

$$G^\nu_\mu = 8\pi T^\nu_\mu$$  \hspace{1cm} (2.38)$$

represent four equations of constraint on the initial data set which is extended to include the quantities $T^\nu_\mu$.

The calculation of the quantities $G^\nu_\mu$ in the $3+1$ formalism first requires the determination of certain components of $R^\nu_\mu$. Specializing the $3+1$ coordinate system by choosing $N=1$ so that the normal, $\vec{n}$, to the hypersurface is the timelike basis vector and using (2.21), the components $^{(4)}R^\nu_\mu$ are determined by

$$^{(4)}R^\nu_{ij\kappa} \tilde{e}_\mu = \nabla_j \nabla_k \tilde{e}_i - \nabla_k \nabla_j \tilde{e}_i$$  \hspace{1cm} (2.39)$$

Now, from (2.34) and (2.35)

$$\nabla_j \nabla_k \tilde{e}_i = \nabla_j \left( \left( ^3 \Gamma^5_{ik} \tilde{e}_s - K^s_{ik} \vec{n} \right) \right)$$

$$= \left( ^6 \Gamma^5_{ik} \tilde{e}_s \right) + \left( ^4 \Gamma^5_{ik} \left( ^8 \Gamma^5_{ik} \tilde{e}_m - K^s_{ik} \vec{n} \right) \right)$$

$$- K^s_{ik} \vec{n} + K^s_{ik} K^5_j \tilde{e}_s$$

Thus

$$\nabla_j \nabla_k \tilde{e}_i = \nabla_k \nabla_j \tilde{e}_i = \left[ \left( K_{ik} - K^s_{ik} \Gamma^5_{ik} - K_i^5 \left( ^3 \Gamma^5_{ik} \right) \right) \vec{n} \right]$$

$$+ \left[ \left( K_i K^5_j - K_i K^5_j \right) + \left( ^6 \Gamma^5_{ik} \right) + \left( ^6 \Gamma^5_{ik} \right) \Gamma^5_{ij} + \left( ^6 \Gamma^5_{ik} \right) \Gamma^5_{ij} + \left( ^6 \Gamma^5_{ik} \right) \Gamma^5_{ij} \right] \tilde{e}_s$$
\[
\n\begin{align*}
\n\nabla_j \nabla_k \tilde{e}_i - \nabla_k \nabla_j \tilde{e}_i &= (D_k K_{ij} - D_j K_{ik}) \tilde{e}_i \\

&+ (K_{ik} K^S_j - K_{ij} K^S_k + R_{ijk}^S) \tilde{e}_S \\

(2.40)
\end{align*}
\]

Comparison with (2.39) gives the result

\[
\begin{align*}
(4) R_{ijk}^S &= (3) R_{ijk}^S + K_{ik} K^S_j - K_{ij} K^S_k \\

(4) R^\alpha_{ijk} &= D_k K_{ij} - D_j K_{ik} \\

(2.41a)
\end{align*}
\]

The desired components of the Einstein tensor may now be calculated using the relation (MTW, equation 14.6)

\[
G_\alpha^\beta = -\delta^{\alpha\mu\nu} \delta_{\beta\tau\sigma} R^{\tau\sigma}_{\mu\nu} \\

(2.42)
\]

where \( \delta^{\alpha\mu\nu} \delta_{\beta\tau\sigma} \) is a permutation tensor having components

\[
\delta^{\alpha\mu\nu} \delta_{\beta\tau\sigma} = \begin{cases} 
+1 & \text{if } \alpha\mu\nu \text{ is an even permutation of } \beta\tau\sigma \\
-1 & \text{if } \alpha\mu\nu \text{ is an odd permutation of } \beta\tau\sigma \\
0 & \text{otherwise}
\end{cases}
\]

and \( |\alpha\beta| \) denotes the restriction \( \alpha > \beta \). Then

\[
\begin{align*}
\nG_0^0 &= -(4) R^{12}_{12} + (4) R^{23}_{23} + (4) R^{31}_{31} \\

&- G_0^0 = (3) R^{12}_{12} + (3) R^{23}_{23} + (3) R^{31}_{31} \\

&- (K^2 K^1_1 - K^2_2 K^1_1 + K^3_2 K^2_2 - K^3_3 K^2_2 + K^1_3 K^3_3 - K^1_1 K^3_3) \\

(2.43)
\end{align*}
\]
Now, the Ricci scalar in three dimensions is

\[ R^{(3)} = R^{i,j}_{i,j} = R^{i,j}_{i,j} \]

where the property \( R^{i,j}_{k\ell} = - R^{j,i}_{k\ell} = + R^{j,i}_{k\ell} \) has been used. Defining

\[ T_{rK} = \delta_{ij} K^{ij} \]

and noting that

\[ (T_{rK})^2 - K_{ij} K^{ij} = (K^{ii})^2 - K_{ij} K^{ij} = 2 \left( k_i^2 + k_j^2 + k_k^2 - k_i^2 k_j^2 - k_k^2 k_i k_j \right) \]

allows (2.43) to be written as

\[ -2G^o_o = R^{(3)} + (T_{rK})^2 - K_{ij} K^{ij} \]

The field equation \( G^o_o = 8\pi T^o_o \) relates the above expression to the local energy density, \( \rho \), as measured by an observer instantaneously at rest in the hypersurface. Thus

\[ -2G^o_o = -16\pi T^o_o = 16\pi \rho \]

\[ R^{(3)} + (T_{rK})^2 - K_{ij} K^{ij} = 16\pi \rho \]

The other required components of the Einstein tensor, \( G \), may be calculated in a similar fashion using (2.42).
For example
\[ G^0_1 = (4) R^{02}_{12} + (4) R^{03}_{13} \]
\[ = D_2 K^2_i - D_i K^2_2 + D_3 K^3_i - D_i K^3_3 \]
\[ = D_i K^i + D_2 K^2_2 + D_3 K^3_i - D_i K^i_2 - D_i K^2_i - D_i K^3_i \]
\[ = D_i K^i - D_i (\text{Tr} K) \]
or, in general
\[ G^0_1 = D^i_j K^i_j - D^i (\text{Tr} K) \quad (2.48) \]

These expressions are to be equated to \( 8n T^0_1 \), which is \( 8n \) times the momentum density, \( j^i \), again as measured by an observer momentarily resting in the hypersurface. Thus
\[ D^i_j K^i_j - D^i (\text{Tr} K) = 8n j^i \quad (2.49) \]

Equations (2.47) and (2.49) are the initial value equations for general relativity. They depend only on quantities definable on a single hypersurface and are coordinate independent in the sense that all of the quantities involved will transform as tensors under any relabelling of the spacetime which leaves the \( t=\text{constant} \) surfaces unchanged.

The six remaining field equations \( G^{i\mu} = 8n T^{i\mu} \) govern the evolution of the initial data \( (g_{ij}^0, K^i_j, \rho, j^i) \) but are not easily determined by means of the process used above for the initial value equations. Arnowitt, Deser and Misner [1] derived the evolution equations (for the vacuum case) by writing the usual action for general relativity.
\[\mathcal{H} = \int \mathcal{L}(x^\mu) \, d^4x^\mu = \int \sqrt{-g} \, (4) \, R \, d^4x^\mu\]  

(2.50)

in the 3+1 formalism and demanding that this integral be stationary with respect to independent variations of the dynamical variables. These were taken to be the six functions \((\mathfrak{g})_{ij}\) and the ADM conjugate momenta \(\pi^{ij}\) closely related to the components of the extrinsic curvature

\[
\pi^{ij} = \sqrt{\det g} \left( g^{ij} \, \nabla^r \mathcal{K} - \mathcal{K}^{ij} \right)
\]

(2.51)

where \(\det g\) is the determinant of the 3-metric. The result is a set of 12 coupled equations involving first time derivatives of \((\mathfrak{g})_{ij}\) and \(\pi^{ij}\). The equations are quite complicated and will not be reproduced here since their exact form is of little, if any, relevance to the initial value problem. Moreover, the ADM form of the evolution equations have not been used in recent numerical constructions of spacetimes - the trend being to use the components \(\mathcal{K}_{ij}\) as the conjugate variables to the 3-metric components [33].

One further feature of the ADM analysis sheds light on the nature of the initial value equations. When written in the 3+1 formalism, the Lagrangian \(\mathcal{L}(x^\mu)\) in (2.50) contains terms linear in the lapse and shift functions. By considering the lapse and the three components of the shift to be independent kinematical variables and requiring that (2.50) remain stationary when varied with respect to them, the initial value equations are recovered. In this way, the constraint equations are seen to be a consequence of the four-fold coordinate freedom of general relativity.
2.4 Preservation of the constraints

From the point of view of constructing a spacetime, equations (2.47) and (2.49) represent constraints which must be satisfied in specifying an initial data set \( (^{(3)}g_{ij}, K_{ij} \rho, j) \) for a spacelike hypersurface, \( \Sigma(t) \). Assuming such a set of data has been determined, one can, in principle at least, evolve the data using the remaining Einstein equations. At any later (earlier) time \( t' \), that is on the hypersurface \( \Sigma(t') \), the evolved data \( (^{(3)}g'_{ij}', K'_{ij}' \rho', j' \) will also satisfy the constraint equations by virtue of the Bianchi identities

\[
G^{\mu\nu} \equiv 0 \quad (2.52)
\]

and the conservation of the stress-energy tensor

\[
T^{\mu\nu} \equiv 0 \quad (2.53)
\]

To see that this is so, the field equations are written in the following form

\[
H^{\mu\nu} \equiv G^{\mu\nu} - 8\pi T^{\mu\nu} \equiv 0 \quad (2.54)
\]

Then the constraint equations are simply

\[
H^{\mu\nu} = H^{\nu\mu} \equiv 0 \quad (2.55)
\]
Now

\[ H^{\mu \nu, \nu} = G^{\mu \nu, \nu} - 8 \pi T^{\mu \nu, \nu} = 0 \quad (2.56) \]

so

\[ H^{\mu \nu, \nu} = H^{\mu \nu, \nu} + (^4 \Gamma^\rho_{\mu \nu} H^{\rho \nu} + ^{^4 \Gamma^\nu_{\rho \nu} H^{\mu \rho}} = H^{\mu, 0} + H^{\mu, i} + (^4 \Gamma^\mu_{\rho \nu} H^{\rho \nu} + ^{^4 \Gamma^\nu_{\rho \nu} H^{\mu \rho}} \quad (2.57) \]

or

\[ H^{\mu, 0} = - (H^{\mu, i} + (^4 \Gamma^\mu_{\rho \nu} H^{\rho \nu} + ^{^4 \Gamma^\nu_{\rho \nu} H^{\mu \rho}}) \quad (2.58) \]

The evolution equations are

\[ H^{\mu \nu} = 0 \quad (2.59) \]

Furthermore, on any hypersurface, the spatial derivatives of \( H^{\mu \nu} \) will also vanish

\[ H^{\mu \nu, i} = 0 \quad (2.60) \]

Equations (2.54), (2.58) and (2.60) now imply that

\[ H^{\mu, 0} = 0 \quad (2.61) \]

with the consequence that the constraint equations hold for the complete evolution, once they have been satisfied on a
initial slice. This point is not too crucial for the initial value problem itself, but is extremely important in the context of numerical evolution of the field equations, where by current methods, constrained data is not automatically evolved into constrained data. In essence, the discrete forms of the field equations which are currently used in numerical relativity are overdetermined systems. Piran considers this overdeterminism to be the most important problem currently facing numerical relativists [32].

This completes the description of the origins and basic nature of the initial value problem. The next section of this chapter is devoted to the description of a particular approach to the initial value problem, chiefly due to York and various collaborators [11], [29], [39], [40], [41].

2.5 York's approach to the initial value problem

York's analysis of the initial value problem is based on a particular way of separating the initial data into freely specifiable and determined-from-constraint pieces. Part of the motivation for his approach was the desire to identify the "true" dynamical gravitational variables - those which correspond to the radiative degrees of freedom - in a covariant manner. From the viewpoint of constructing initial data, however, the chief success of York's work is that it puts the initial value equations into a form which allows them to be solved, at least in principle, with existing numerical methods and in some cases, by analytic means. (Since this
section deals entirely with 3-dimensional quantities, the superscript \(^4\) will no longer be employed.

The simplest part of York's prescription for the specification of initial data is to give freely on the hypersurface the value of the mean extrinsic curvature, \(\text{Tr}K\)

\[
\text{Tr}K \equiv g_{ij} K^{ij} \tag{2.45}
\]

There are both physical and practical motivations for choosing to do this. From the physical point of view, \(\text{Tr}K\) provides a measure of the local change in volume of the hypersurface as it evolves into a nearby slice. For example, if \(\text{Tr}K\) vanishes, then the hypersurface has maximal volume, as is the case for the standard \(t=\text{constant}\) hypersurfaces of Minkowskii spacetime. In addition, the value of \(\text{Tr}K\) is very sensitive to the way in which the hypersurface is embedded in the spacetime - a localized deformation (evolution) of the slice will cause \(\text{Tr}K\) to change somewhere in the neighborhood of the deformation. In this sense, \(\text{Tr}K\) has been interpreted as a natural, local "time" function [23], [29]. Also, certain simple choices of \(\text{Tr}K\) lead to significant simplification of the initial value equations as will be shown later in this section. In the following analysis, then, \(\text{Tr}K\) will be assumed to be given, so it is convenient to isolate the trace-free part of the extrinsic curvature \(K^{ij}\) as follows

\[
E^{ij} = K^{ij} - \frac{1}{2} g^{ij} \text{Tr}K
\]

\[
g_{ij} E^{ij} = 0
\]

\[
K^{ij} = E^{ij} + \frac{1}{2} g^{ij} \text{Tr}K \tag{2.62b}
\]
A key idea in York's program which is originally due to Lichnerowicz is to specify the initial data \((g_{ij}, E^i, trK, \rho, j)\) only up to a conformal transformation. For example, the 3-metric is written as

\[
g_{ij} = \Psi^4 \hat{g}_{ij} \tag{2.63}
\]

where \(\Psi\) is a strictly positive function of position on the hypersurface, and the base metric, \(\hat{g}_{ij}\) is assumed to be given. If \(V^i\) and \(W^i\) are vectors tangent to the hypersurface at the same point, then the angle \(\theta\) between them

\[
\theta = \cos^{-1} \left( \frac{g_{ij} V^i W^j}{(g_{ij} V^i V^j g^{k\lambda} W^k W^\lambda)^{\frac{1}{2}}} \right) \tag{2.64}
\]

is clearly unchanged under such a transformation - hence the term "conformal". Physically, the conformal factor determines the local length scale of the surface.

Associated with \(\hat{g}_{ij}\) is its inverse \(\hat{g}^{ij}\) which satisfies

\[
\hat{g}^{ij} \hat{g}_{jk} = \delta^{i}_{\kappa} \tag{2.65}
\]

Another useful relation is

\[
\sqrt{g} = \Psi^6 \sqrt{\hat{g}} \tag{2.66}
\]
It is seen from (2.64) and (2.66) that the quantity

\[ q^{-\frac{1}{2}} q_{ij} \]

remains invariant under a conformal transformation. A covariant derivative \( \hat{D}_i \) compatible with \( \hat{g}_{ij} \) may be introduced.

\[ \hat{D}_i \hat{g}_{jk} = \hat{D}_i \hat{g}^{jk} = 0 \quad (2.67) \]

The connection components \( \hat{\Gamma}^i_{jk} \) are easily shown to be related to \( \hat{\Gamma}^i_{jk} \) as follows

\[ \hat{\Gamma}^i_{jk} = \hat{\Gamma}^i_{jk} + 2 \Psi^{-4} (\delta^i_k \Psi_{,j} + \delta^i_j \Psi_{,k} - \hat{g}^{im} \hat{g}_{jk} \Psi_{,m}) \quad (2.68) \]

In addition, a rather lengthy but straightforward calculation gives the following relationship between the scalar curvatures of the physical and base metrics

\[ R = \Psi^{-4} \hat{R} - 8 \Psi^{-5} \hat{\Delta} \Psi \quad (2.69) \]

where \( \hat{\Delta} \) is the Laplacian in the base metric

\[ \hat{\Delta} = \hat{D}_i \hat{D}^i \quad (2.70) \]

The other initial data, with the exception of \( \text{Tr}K \), are also to be conformally specified. The trace-free part of the extrinsic curvature is transformed as follows

\[ E^{ij} = \Psi^{-10} \hat{E}^{ij} \quad (2.71) \]
Clearly, $E$ is also trace-free. Moreover, this particular scaling gives the following result:

$$D_j E^{ij} = \Psi^{-10} \hat{D}_j \hat{E}^{ij} \quad (2.72)$$

since

$$D_j E^{ij} = E^{ij}_{\ j} + \Gamma^{i}_{k} E^{kj} + \Gamma^{k}_{ij} E^{ji}$$

$$= (\Psi^{-10} \hat{E}^{ij}), \ j +$$

$$(\hat{\nabla}_{i} E^{ij} + 2 \Psi^{-10} (\delta^{i}_{j} \Psi, \kappa + \delta^{j}_{i} \Psi, \chi - \hat{g}^{iz} \hat{g}_{kj} \Psi, \lambda) \Psi^{-10} \hat{E}^{ij} +$$

$$(\hat{\nabla}_{j} E^{ji} + 2 \Psi^{-10} (\delta^{i}_{j} \Psi, \chi + \delta^{j}_{i} \Psi, \kappa - \hat{g}^{iz} \hat{g}_{kj} \Psi, \lambda) \Psi^{-10} \hat{E}^{ji})$$

$$= -10 \Psi^{-10} \hat{E}^{ij} + \Psi^{-10} (E^{ij}_{\ j} + \hat{\nabla}^{i}_{k} \hat{E}^{kj} + \hat{\nabla}^{k}_{ij} \hat{E}^{ji}) +$$

$$2 \Psi^{-10} \left[ (\delta^{i}_{j} \Psi, \kappa \hat{E}^{ji} + \delta^{j}_{i} \Psi, \kappa \hat{E}^{ij} - \Psi, \lambda \hat{g}^{iz} \hat{g}_{kj} \Psi, \lambda \hat{E}^{ji}) +$$

$$3 \Psi^{-10} \hat{E}^{ij} \right]$$

$$= \Psi^{-10} \hat{D}_j \hat{E}^{ij}$$

Finally, the quantities $\hat{\rho}$ and $\hat{j}^{i}$ are assumed to be specified so that

$$\rho = \Psi^{-6} \hat{\rho} \quad (2.73)$$

and

$$j^{i} = \Psi^{-10} \hat{j}^{i} \quad (2.74)$$

are the physical energy and momentum densities respectively. York has justified the scaling factors used in (2.73) and (2.74) through dimensional considerations, but perhaps more importantly, if the conformal data satisfies the dominant energy condition as given by Hawking and Ellis, [20], then so
will the physical data, regardless of the exact nature of $\Psi$. The dominant energy condition asserts that the local 4-momentum density

$$T^\mu = \dot{T}^\mu \eta_\nu$$  \hspace{1cm} (2.75)

is a non-spacelike vector. For the conformal stress-energy tensor $\dot{T}^{\mu\nu}$ this implies

$$\dot{T}^\mu \dot{T}_\mu - q_{ij} \ddot{\jmath}^i \ddot{\jmath}^j - \rho^2 < 0$$  \hspace{1cm} (2.76)

or

$$\rho > \left( q_{ij} \ddot{\jmath}^i \ddot{\jmath}^j \right)^{1/2}$$  \hspace{1cm} (2.77)

Thus if $\dot{\rho}$ and $\ddot{\jmath}$ are chosen to satisfy the above, then the scaling given by expressions (2.73) and (2.74) guarantees that

$$\rho > \left( q_{ij} \ddot{\jmath}^i \ddot{\jmath}^j \right)$$  \hspace{1cm} (2.78)

The second major idea in York's approach to the initial value problem is to decompose the trace-free part of the conformal extrinsic curvature into transverse-traceless (TT) and longitudinal parts - a process which can be performed on any symmetric trace free tensor [39]. This procedure is analogous to what may be done in the study of the initial value problem for electrodynamics. There, the electric field $E$
may be split into transverse, $\tilde{E}_T$, and longitudinal, $\tilde{E}_L$, parts as follows

$$\tilde{E} = \tilde{E}_L + \tilde{E}_T$$

$$\nabla \cdot \tilde{E}_T = 0$$

$$\tilde{E}_L = \nabla U$$  \hspace{1cm} (2.79)

where $U$ is a scalar function. The equation of constraint for the electric field

$$\nabla \cdot \tilde{E} = 4\pi \rho_e$$  \hspace{1cm} (2.80)

where $\rho_e$ is the charge density becomes a second order elliptic equation for the function $U$

$$\nabla^2 U = 4\pi \rho_e$$  \hspace{1cm} (2.81)

which is the single initial value equation for electrodynamics. The transverse field $\tilde{E}_T$ may be freely specified and represents the radiative degrees of freedom of the electromagnetic field (see [41] for more details).

In a similar fashion, the transverse-traceless/longitudinal decomposition of $\hat{E}^{ij}$ separates the freely specifiable part of $\hat{E}^{ij}$ from the part which will be determined from the constraint equations (2.47) and (2.49). Thus, put

$$\hat{E}^{ij} = \hat{E}_{TT}^{ij} + \hat{E}_{T}^{ij}$$  \hspace{1cm} (2.82)
where the longitudinal part, $\hat{E}_{\ell}^{ij}$ is derived from differentiation of some vector $W^i$ as follows

$$\hat{E}_{\ell}^{ij} = \hat{D}^i W^j + \hat{D}^j W^i - \frac{2}{3} q^{ij} \hat{D}_k W^k \quad (2.83)$$

This guarantees that $\hat{E}_{\ell}^{ij}$ is both symmetric and tracefree

$$\hat{E}_{\ell}^{ij} = \hat{E}_{ij}$$

$$\hat{g}_{ij} \hat{E}_{\ell}^{ij} = 2 \hat{D}_j W^j - \frac{2}{3} \delta^{ij} \hat{D}_k W^k = 0$$

Now from (2.82)

$$\hat{E}_{TT}^{ij} = \hat{E}_{ij} - \hat{E}_{\ell}^{ij} \quad (2.84)$$

The tracefree condition on $\hat{E}_{TT}^{ij}$ is satisfied by construction

$$\hat{g}_{ij} \hat{E}_{TT}^{ij} = \hat{g}_{ij} \hat{E}_{ij} - \hat{g}_{ij} (\hat{\lambda} W)^i = 0$$

and the transversality requirement

$$\hat{D}_j \hat{E}_{TT}^{ij} = 0 \quad (2.85)$$

leads to the following equation for the vector $W^i$

$$\hat{D}_j (\hat{\lambda} W)^{ij} \equiv (\hat{\Delta} \lambda W)^i = \hat{D}_j \hat{E}^{ij} \quad (2.86)$$
The operator $\hat{\Delta}_\lambda$ defined above, is called the "vector Laplacian". Its explicit form, using (2.83) is

\[
(\hat{\Delta}_\lambda W)^i = \hat{D}_j \hat{D}^i W^j + \hat{D}_j \hat{D}^i W^j - \frac{3}{2} \hat{D}^i \hat{D}_k W^k
\]

where a form of the Ricci identity has also been used. The properties of $\hat{\Delta}_\lambda$ are examined in depth in [11] and [39] along with conditions necessary for existence and uniqueness of a solution $W^i$ of equation (2.86).

The transverse-traceless part of $E^{ij}$ may also be decomposed into a freely specifiable symmetric, tracefree part $\hat{T}^{ij}$ and a longitudinal part $-(\hat{\lambda} V)^{ij}$. Thus write,

\[
\hat{E}^{ij} = \hat{T}^{ij} - (\hat{\lambda} V)^{ij}
\]

Again, the transversality of $\hat{E}^{ij}$ demands that

\[
(\hat{\Delta}_\lambda V)^i = \hat{D}_j \hat{T}^{ij}
\]

Using equations (2.82), (2.83) and (2.88), the trace-free part of the conformally scaled extrinsic curvature is

\[
\hat{E}^{ij}_\lambda = \hat{T}^{ij} - (\hat{\lambda} V)^{ij} + (\hat{\lambda} W)^{ij}
\]

where

\[
X^i = W^i - V^i
\]
The initial value equations may now be written in terms of the conformally scaled data and the functions $\Psi$ and $X^i$. Recall equation (2.47)

\[ R + (Tr K)^2 - K^{ij} K_{ij} = 16\pi \rho \] (2.47)

Using equations (2.62), (2.69), (2.71), (2.73) and (2.90), this becomes

\[ \Psi^{-4} \hat{R} - 8 \Psi^{-5} \hat{\Delta} \Psi + (Tr K)^2 - \Psi^{-2} \left( \hat{T}^{ij} + (\hat{\ell} X)^{ij} \right) \left( \hat{T}_{ij} + (\hat{\ell} X)_{ij} \right) - \frac{1}{3} (Tr K)^2 \]

\[ = 16\pi \rho \Psi^{-6} \]

or

\[ -8 \hat{\Delta} \Psi = -\hat{R} \Psi - \frac{2}{3} (Tr K)^2 \Psi^5 + \left[ \hat{T}^{ij} + (\hat{\ell} X)^{ij} \right] \Psi^3 + 16\pi \rho \Psi^{-3} \] (2.92)

The other three initial value equations,

\[ D_j K^{ji} - D^i (Tr K) = 8\pi j^i \] (2.49)

may be rewritten using (2.62), (2.72), (2.74) and (2.90)

\[ \Psi^{-10} \hat{D}_j \left( \hat{T}^{ij} + (\hat{\ell} X)^{ij} \right) + \frac{1}{3} \Psi^4 \hat{D}^i Tr K - \Psi^4 \hat{D}^i Tr K = 8\pi \Psi^{-10} \hat{j}^i \]

or

\[ \left( \hat{\Delta}_\ell X \right)^i = -\hat{D}_j \hat{T}^{ij} + \frac{3}{2} \Psi^6 \hat{D}^i Tr K + 8\pi \hat{j}^i \] (2.93)
In summary, the freely specifiable quantities in this approach to the initial value problem are 1) the base 3-metric $\hat{g}_{ij}$; 2) the symmetric trace-free tensor $\hat{T}^{ij}$; 3) the mean extrinsic curvature $\text{Tr}K$, and 4) the conformally scaled energy and momentum densities $\hat{\rho}$ and $\hat{j}^i$. Once these quantities have been specified, equations (2.92) and (2.93) are to be solved for $\Psi$ and $X^i$. The physical initial data is then determined using the following expressions

\begin{align*}
\hat{g}_{ij} &= \Phi^4 \hat{g}_{ij} \\
K^{ij} &= \Phi^{-6} (\hat{\nabla}^i \hat{\xi}^j + (\hat{\xi} \hat{\xi})^{ij}) + \frac{1}{3} \Phi^{-4} g_{ij} \text{Tr}K \\
\rho &= \Phi^{-6} \hat{\rho} \\
\hat{j}^i &= \Phi^{-10} \hat{j}^i
\end{align*} \hspace{1cm} (2.94)

In general, (2.92) and (2.93) represent a system of four coupled quasi-linear (linear in the second derivatives) elliptic partial differential equations. In the special case of constant mean extrinsic curvature, however, (2.93) decouples from (2.92) and also becomes linear.

Physically, the four functions $\Psi$ and $X^i$ may be interpreted as generalizations of the single potential, $\phi$, of Newtonian gravitation. In this interpretation, the initial value equations constitute the relativistic analogue of the Poisson equation which relates the Newtonian potential to the mass density $\rho_M$

\begin{equation}
\nabla^2 \phi = 4 \pi \rho_M \hspace{1cm} (2.95)
\end{equation}
The questions of existence and uniqueness of solutions of the initial value equations have been examined by several researchers. Here, topological considerations play an important role since the constraint equations do not determine the topology of the initial slice. York's original work on the TT-decomposition of symmetric tensors was restricted to closed positive definite manifolds. Conditions for existence and uniqueness in this case are discussed in depth in reference [11].

The other major topological class of initial data which has been examined is that of asymptotically flat slices. Roughly speaking, asymptotic flatness means that the initial data \((g_{ij}, K^{ij})\) approaches that of a standard \(t=\)constant slice of Minkowski spacetime at large distances from an arbitrarily chosen origin on the slice. In addition, the approach is assumed rapid enough to ensure that the total energy and momentum of the spacetime are well defined and conserved quantities. Denoting a flat 3-metric by \(f_{ij}\), and letting \(r\) be the distance from some origin with respect to \(f_{ij}\), then the following is a typical definition of asymptotic flatness [40]

\[
g_{ij} = f_{ij} + h_{ij}
\]

\[
\lim_{r \to \infty} h_{ij} = \lim_{r \to \infty} (f^{ik} h_{kj}) = O\left(\frac{1}{r}\right)
\]

\[
\lim_{r \to \infty} K_{ij} = O\left(\frac{1}{r^2}\right)
\]

(2.96)

In the case of asymptotically flat initial data, it can
be shown [29] that the asymptotic behaviour of \( \Psi \) determines
the total energy of the spacetime. For this reason, equation
(2.92) is often called the Hamiltonian constraint. Similarly
the behavior of the "vector potential", \( X^i \), at large distances
measures the total momentum of the spacetime, and equation
(2.93) is therefore termed the momentum constraint. The
conditions for existence and uniqueness of solutions of the
constraints have not been determined as completely for
asymptotically flat slices as they have for closed slices.
Some known results are summarized in [11].

2.6 Past work on initial value problems

This section presents a brief review of some of the
initial value problems which have been studied to this date.

A rather special case of the initial value problem
results when the initial slice is assumed to have vanishing
extrinsic curvature

\[ K_{ij} = 0 \]

Such a slice is often called time-symmetric. In this case, the
momentum constraints are automatically satisfied and the
Hamiltonian constraint (2.92) becomes

\[ -8 \hat{\Delta} \Psi = - \hat{\mathcal{R}} \Psi + 16 \pi \rho \Psi^{-3} \]  (2.97)

If attention is further limited to the vacuum case, the above
becomes

\[ \hat{\Delta} \Psi + \frac{1}{8} \hat{\mathcal{R}} \Psi = 0 \]  (2.98)
Brill [9] studied this equation for a rather unphysical, but exactly soluble, case involving an axially symmetric, "square well" scalar curvature. The solutions determined indicated the presence of gravitational radiation ("time-symmetric Brill waves") having a total energy roughly proportional to the square of the amplitude of the radiation. In addition, it was found that the data described a black hole when the amplitude was sufficiently large. Eppley [14] solved a similar problem numerically using a more physical curvature scalar and verified the qualitative features of Brill's analysis. He then extended his study to the case of non-time-symmetric data [15], in which two of the momentum constraints, as well as the Hamiltonian constraint, were solved numerically.

Misner [26] studied another vacuum time-symmetric problem which was exactly soluble. He demanded that the metric of the interior of the asymptotically flat slice be conformally related to that of a 3-dimensional "doughnut". The solutions Misner constructed described slices having multiply connected topologies and were eventually interpreted as representing black holes. Smarr et al. [35] later used this initial data in their work on the head-on collision of two black holes.

More recently, Nakamura et al. [28] have studied the initial value problem for the case of an axially symmetric collapsing star, which required the numerical solution of both the Hamiltonian constraint and one of the momentum constraints. The resulting data has also been evolved. In addition, there have been numerous other studies recently
whose chief concern is in evolving initial data numerically, rather than generating it. A review article by Piran [32] summarizes the state of these studies as of 1981. All of the problems studied to this date have had symmetries which have allowed the initial value unknowns to be written as functions of, at most, two spatial variables. In addition, most of the problems which have been examined have dealt with asymptotically flat slices.
CHAPTER 3

A Specific Initial Value Problem

Working within the formalism described in the previous chapter, Bowen and York [3] and Piran and York [41] have recently determined solutions to equations (2.92) and (2.93) which have been interpreted as providing initial data for two new families of black holes. Bowen and York hope that their method of attacking the problem will eventually prove useful in producing initial data for a non-head-on collision of two or more black holes - a problem which would appear to be of significant interest in the context of the production of gravitational radiation. Due to the introduction of several simplifying assumptions regarding the nature of the initial slices, Bowen and York were able to find analytic solutions to the momentum constraint equations (2.93). Exact solutions of the Hamiltonian constraint could not be found, so numerical solutions were determined by Piran. The main goal of the current work was to re-solve the Hamiltonian constraint numerically, both to check Piran's results and to investigate a numerical technique which may prove useful in the solution of more complicated initial value problems. This chapter first reviews the work done by Bowen and York in [3] (Sections 3.1 - 3.2). Because the emphasis of this thesis is on numerical
methods, no attempt has been made by the author to reproduce their analytic results. Section 3.3 then describes how the Hamiltonian constraint equations for the new geometries are posed as discrete problems which may be solved numerically.

3.1.1 Topology of the slices and simplifying assumptions

As mentioned in the previous chapter, there are two major topological classes of spacelike hypersurfaces which have received the most attention in the context of the initial value problem - closed slices and asymptotically flat slices. In the present case, the hypersurfaces constructed are asymptotically flat. This allows quantities such as the total energy, linear momentum and angular momentum of the spacetime, as measured at spacelike infinity, to be defined. In addition, the requirement of asymptotic flatness provides a boundary condition for the conformal factor in the solution of the Hamiltonian constraint, as will be seen below.

To simplify the analysis of the initial value equations, Bowen and York restricted attention to maximal volume, conformally flat, vacuum slices. Recall that a maximal volume hypersurface has vanishing mean extrinsic curvature

$$\text{Tr} \ K = 0$$  \hspace{1cm} (3.1)

Conformal flatness means that the physical metric, $g_{ij}$, of the hypersurface may be written as

$$g_{ij} = \gamma^4 f_{ij}$$  \hspace{1cm} (3.2)
where \( f_{ij} \) is the usual 3-dimensional Euclidean metric. Note that this immediately implies that the conformally scaled curvature scalar, \( \hat{R} \), identically vanishes. Finally, the vacuum requirement simply means that there are no matter or momentum sources on the hypersurface

\[
\rho = \psi^{-8} \hat{\rho} = 0 \quad (3.3)
\]

\[
\hat{j}^i = \psi^{-10} \hat{j}^i = 0 \quad (3.4)
\]

### 3.1.2 Solution of the momentum constraint

With the above restrictions, the momentum constraint simplifies to

\[
(\Delta_{\lambda} \chi)^i = - \hat{D}_j \hat{T}^{ij} \quad (3.5)
\]

This equation may be further simplified by demanding that \( \hat{T}^{ij} \), which may be freely specified, also vanish. Together with the vanishing of \( \text{Tr}K \), this implies that the conformally scaled extrinsic curvature, \( \hat{K}^{ij} \) and the physical extrinsic curvature are both purely longitudinal

\[
\hat{K}^{ij} = (\hat{\ell} \chi)^{ij} \quad (3.6)
\]
\[ K^{ij} = \Psi^{-10} \tilde{K}^{ij} \quad (3.7) \]

Equation (3.5) becomes
\[ (\tilde{\Delta}_x X)^i = \tilde{D}_j (A X)^i j = \tilde{D}_j \tilde{K}^{ij} = 0 \quad (3.8) \]

Using (2.87) with \( R = 0 \), and denoting the flat-space Laplacian by \( \nabla^2 \), the explicit form of the above equation is
\[ (\tilde{\Delta}_x X)^i = \nabla^2 X^i + \frac{1}{3} \tilde{D}^i \tilde{D}_j X^j = 0 \quad (3.9) \]

Bowen and York determined solutions to this equation by writing \( X^i \) as
\[ X^i = \nabla^i - \frac{1}{4} \tilde{D}^i \lambda \quad (3.10) \]

They then solved successively the equations
\[ \nabla^2 \nabla^i = 0 \]
\[ \nabla^2 \lambda = \tilde{D}_i \nabla^i \quad (3.11) \]

The reader may easily verify that such a procedure yields vectors \( X^i \) which satisfy (3.9). Of the many possible \( \tilde{K}_{ij} \)'s that could be constructed in this fashion, the following were chosen
\[ \tilde{K}_{ij} = \frac{3}{2r^2} \left[ P_i n_j + P_j n_i - (\delta_{ij} - n_i n_j) P_k n_k \right] \]
\[ = \frac{3a^2}{2r^4} \left[ P_i n_j + P_j n_i + (\delta_{ij} - 5 n_i n_j) P_k n_k \right] \quad (3.12a) \]
Here, $r$ is the Euclidean distance from some arbitrary point of the hypersurface, $n^i$ is the unit normal of a $r=$constant 2-sphere, $P^i$ and $J^i$ are constant vectors, $\varepsilon_{ijk}$ is the permutation tensor and $\alpha$ is a constant. These particular solutions were chosen for two main reasons which will be discussed below.

3.2 The Hamiltonian constraint and inversion techniques

The Hamiltonian constraint (2.92), in the present case, where $\text{Tr}K=\vec{\rho}=\bar{R}=0$, is

$$\nabla^2 \Psi + \frac{(\vec{\nabla}X)^{ij}(\vec{\nabla}X)_{ij}}{8 \Psi^7} = 0$$

or

$$\nabla^2 \Psi + \frac{\hat{K}_{ij} \hat{R}_{ij}}{8 \Psi^7} = 0 \quad (3.13)$$

where $\hat{K}_{ij}$ will be determined by (3.12a) or (3.12b). The requirement that a conformally flat hypersurface be asymptotically flat translates into the following boundary condition on $\Psi$.

$$\lim_{r \to \infty} \Psi = 1 + O\left(\frac{1}{r}\right) \quad (3.14)$$
As it stands, the problem of determining a unique, positive \( \gamma \) which satisfies (3.13) everywhere on the hypersurface, subject to the boundary condition (3.14) is not well-posed since the scaled extrinsic curvatures of (3.12a) and (3.12b) diverge at \( r=0 \). To circumvent this problem, Bowen and York further restricted attention to hypersurfaces which are \textit{isometric} with respect to a mapping through a 2-sphere of radius \( a \). Introducing the usual spherical polar coordinates \((r, \theta, \phi)\) on the slice, the mapping has the explicit form

\[
\begin{align*}
    r' &= \frac{a^2}{r}; \quad \theta' = \theta; \quad \phi' = \phi
\end{align*}
\]  

(3.15)

or in Cartesian coordinates \((x,y,z)\)

\[
\begin{align*}
    x' &= \frac{a^2}{r^2} x; \quad y' &= \frac{a^2}{r^2} y; \quad z' &= \frac{a^2}{r^2} z
\end{align*}
\]  

(3.16)

This transformation is defined for \( x' \in \mathbb{R}^3\setminus\{0\} \) and maps the region \( 0<r<a \) onto \( r>a \) and vice versa. Now a map

\[
    x'^i = x'^i(x^i)
\]  

(3.17)

is an isometry of a metric \( \tilde{g}_{ij} \) if

\[
\tilde{g}_{ij}(x^i) = \frac{\partial x^k}{\partial x^i} \frac{\partial x^l}{\partial x^j} \tilde{g}_{kl}(x^i)
\]  

(3.18)

In the present case, the physical metric \( g_{ij} \) is conformally flat. If \( x^i \) are Cartesian coordinates, then

\[
g_{ij}(x^i) = \gamma^4(x^i) \delta_{ij}
\]  

(3.19)
and for the transformation of (3.16)

$$\frac{\partial x'_{i}}{\partial x_{j}} = \delta_{j}^{i} \frac{a^{2}}{r^{2}}$$ \hspace{2cm} (3.20)

Thus (3.16) is an isometry of \( g_{ij} \) if

$$g_{ij}(x^{'i}) = \left( \delta_{i}^{k} \frac{a^{2}}{r^{2}} \right) \left( \delta_{j}^{l} \frac{a^{2}}{r^{2}} \right) g_{kl}(x^{'i})$$

or

$$\Psi^{4}(x^{'i}) \delta_{ij} = \frac{a^{4}}{r^{4}} \Psi^{4}(x^{'i}) \delta_{ij}$$

which immediately implies

$$\Psi(x^{'i}) = \frac{a}{r} \Psi(x^{'i})$$ \hspace{2cm} (3.21)

Differentiation of the last expression with respect to \( r \) gives

$$\frac{\partial \Psi(x^{'i})}{\partial r} = - \frac{a}{r^{2}} \Psi(x^{'i}) + \frac{a}{r} \frac{\partial r'}{\partial r} \frac{\partial \Psi(x^{'i})}{\partial r'}$$

$$= - \frac{a}{r^{2}} \Psi(x^{'i}) - \frac{a^{3}}{r^{3}} \frac{\partial \Psi(x^{'i})}{\partial r'}$$ \hspace{2cm} (3.22)

Evaluating this result when \( r(x^{'i}) = r'(x^{'i}) = a \) yields

$$\frac{\partial \Psi}{\partial r} + \left. \frac{\Psi}{2r} \right|_{r=a} = 0$$ \hspace{2cm} (3.23)

Mathematically, this equation provides a second boundary condition for the conformal factor which allows one to avoid having to deal with the singular behavior of the scaled extrinsic curvature at the origin. Geometrically, the process of imposing an isometry condition on the metric of the slice in the above fashion leads to the identification of the
two regions \(0<r\leq a\); \(r>a\) as two separate, but identical, asymptotically flat sheets, joined in a smooth fashion at \(r=a\). Such a description is often used to characterize the geometry of the time-symmetric \((K_{\hat{ij}}=0)\) hypersurfaces of the Schwarzschild spacetime (see for example [27], p.836) In this case, the "throat" which connects the two asymptotically flat sheets coincides with the apparent horizon [20] of the Schwarzschild black hole. Bowen and York concluded that the slices they were constructing could also be interpreted as hypersurfaces of spacetimes containing black holes. Specifically, it was argued that a slice having the scaled extrinsic curvature given by (3.12a) would represent a "snapshot" of a boosted black hole having linear momentum \(P^i\) as measured by an observer at spacelike infinity, and a hypersurface with the \(\hat{K}_{ij}\) of (3.12b) would correspond to a spinning black hole having intrinsic angular momentum \(J^i\). The ability to interpret the results in this fashion provided part of the motivation for choosing the particular solutions of the momentum constraints. In addition, it was shown that the \(\hat{K}_{ij}\)'s of (3.12) have the property that if a conformal factor \(\gamma(r, \theta, \phi)\) is found which satisfies (3.13) in the region \(r>a\) subject to the boundary conditions (3.14), and (3.23), then the function

\[
\gamma' \left( r', \theta', \phi' \right) = \frac{a}{r} \gamma \left( \frac{a^2}{r}, \theta, \phi \right)
\]

automatically satisfies (3.13) in the region \(0<r\leq a\). That is, the \(\hat{K}_{ij}\)'s of (3.12a) and (3.12b) were constructed so that the physical extrinsic curvature would also be invariant under the
inversion transformation. (Actually, since the Hamiltonian constraint is quadratic in \( \hat{K}_{ij} \), a change of sign in \( \hat{K}_{ij} \) under inversion is allowed. The "+" solution given by (3.12a) transforms without change in sign; the "-" solution transforms with a sign change. Bowen and York interpret the latter case as indicating that a momentum of \(-P^r\) is associated with the "bottom" sheet of the slice (0<\(r\leq a\)), if momentum \(+P^r\) is associated with the "top" sheet (\(r>a\)).

3.3 Discretization of the Hamiltonian constraint

With the inversion technique described in the previous section, the problem of solving the Hamiltonian constraint becomes the problem of solving the following boundary value problem

\[
\nabla^2 \psi + \frac{\hat{K}_{ij} \hat{K}^{ij}}{8\phi^2} = 0 \quad r > a
\]

(3.13)

\[
\lim_{r \to \infty} \psi = 1
\]

(3.14)

\[
\left. \frac{\partial \psi}{\partial r} + \frac{\psi}{2r} \right|_{r=a} = 0
\]

(3.23)

with \( \hat{K}_{ij} \) given by (3.12a) or (3.12b). The purpose of this section is to describe the manner in which this problem is converted to a form suitable for solution by numerical means.

An examination of the explicit forms of (3.12a) and (3.12b) reveals that the associated boundary value problems will all have symmetries which can be exploited in their
subsequent numerical solution. Spherical coordinates \((r, \theta, \phi)\), with the usual relation to Cartesian coordinates \((x, y, z)\) will be employed.

The (flat) metric in spherical coordinates is

\[
f_{ij} = \text{diag} (1, r^2, r^2 \sin^2 \theta) \tag{3.24}
\]

In these coordinates, a unit normal, \(n^i\) to a 2-sphere centred at \(r=0\) has components \((1, 0, 0)\). Consider expression (3.12a) first and take \(P^i\) in the +z direction. Thus

\[
P^i = \left( P \cos \theta, -\frac{P \sin \theta}{r}, 0 \right) \tag{3.25}
\]

\[
P = \left( \frac{1}{2} f_{ij} P^i P^j \right)^{1/2}
\]

Then, the only non-vanishing components of \(\hat{K}^+_{ij}\) are

\[
\hat{K}^\parallel_{rr} = \frac{3P \cos \theta}{r^2} \left[ 1 + \frac{a^2}{r^2} \right]
\]

\[
\hat{K}^\parallel_{\theta \theta} = -\frac{3P \cos \theta}{2} \left[ 1 + \frac{a^2}{r^2} \right]
\]

\[
\hat{K}^\parallel_{\phi \phi} = -\frac{3P \cos \theta \sin^2 \theta}{2} \left[ 1 + \frac{a^2}{r^2} \right]
\]

\[
\hat{K}^\parallel_{r \theta} = \hat{K}^\parallel_{\theta r} = -\frac{3P \sin \theta}{2r} \left[ 1 + \frac{a^2}{r^2} \right]
\]

Denoting \(\hat{K}^\parallel_{ij}\) by \(H^+_{\phi} \) (B \rightarrow "boosted"), then

\[
H^+_{\phi} = \hat{K}^\parallel_{ij} \hat{K}^\parallel_{ji}
\]

\[
= \hat{K}^\parallel_{rr} + \frac{2}{r^2} \hat{K}^\parallel_{\theta \theta} + \frac{1}{r^4} \hat{K}^\parallel_{\phi \phi} + \frac{1}{r^4 \sin^2 \theta} \hat{K}^\parallel_{r \phi}
\]

\[
= \frac{9}{2} \frac{P^2}{r^4} \left[ (1 + \frac{a^2}{r^2})^2 + 2 \cos \theta (1 + \frac{4a^2}{r^2} + \frac{a^4}{r^4}) \right] \tag{3.26}
\]

Similarly, taking \(J^i\) in the +z direction, the only non-
vanishing component of expression (3.12b) is
\[ \hat{K} r \phi = -\frac{3P \sin^4 \theta}{r^2} \]
so \( H^2 (S \rightarrow \text{"spinning"}) \) is
\[ H^2 = \frac{2}{r^2 \sin^2 \theta} \hat{K} r \phi = \frac{18 J^2 \sin^2 \theta}{r^6} \quad (3.27) \]

Since neither \( H^2 \) or \( H^4 \) has any \( \phi \) dependence, the solutions of (3.13) will be axially symmetric (symmetric about the z-axis), as should be expected, since the vectors \( P \) and \( J \) were taken along the z-axis. In addition, since \( \cos^2(\theta) = \cos^2(\pi - \theta) \) and \( \sin^2(\theta) = \sin^2(\pi - \theta) \) the solutions will also possess a reflection symmetry about the \( \theta = \frac{\pi}{2} \) axis. Thus, it will suffice to solve (3.13) on a domain such as that shown in Figure 3. The symmetries provide the following additional boundary conditions for \( \psi \)
\[ \frac{\partial \psi}{\partial \theta} \bigg|_{\theta = 0} = 0 \quad (3.28) \]
\[ \frac{\partial \psi}{\partial \theta} \bigg|_{\theta = \frac{\pi}{2}} = 0 \quad (3.29) \]

The unboundedness of the domain presents a problem in the solution of (3.13) by numerical means, since a computational domain must be finite. York and Piran handled this difficulty by using the known asymptotic behavior of \( \psi \) to produce an approximate boundary condition which could be imposed at a finite radius. This scheme was originally used in the present work until it was suggested by Gus Gassmann that the infinite
Figure 3
Domain of Boundary Value Problem
domain be transformed to a finite domain in the following fashion. Introduce a new radial coordinate \( s \), defined by

\[
s = 1 - \frac{a}{r}
\]  

(3.30)

This takes the infinite domain \( \{ P(r, \theta) \mid a < r < \infty ; 0 < \theta < \frac{\pi}{2} \} \) into the bounded domain \( \{ P(s, \theta) \mid 0 < s < 1 ; 0 < \theta < \frac{\pi}{2} \} \). The use of this coordinate change proved to be quite advantageous numerically.

The Laplacian of \( \nabla \) in \((s, \theta)\) coordinates is

\[
\nabla^2 \nabla = \frac{(1-s)^2}{a^2} \frac{\partial^2}{\partial s^2} \left( \sin \theta \frac{\partial \psi}{\partial \theta} \right) + \frac{(1-s)^2}{a^2 \sin \theta} \frac{2}{\theta} \left( \sin \theta \frac{\partial \psi}{\partial \theta} \right)
\]  

(3.31)

This expression is singular when \( s=1 \) or when \( \theta = 0 \). The behavior at \( s=1 \) presents no problem numerically, since as will be seen below, the discrete approximation of the operator does not have to be evaluated at \( s=1 \). However, the discrete form of

\[
\frac{1}{\sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial \psi}{\partial \theta} \right)
\]

must be evaluated at \( \theta = 0 \). Thus, at \( \theta = 0 \) the above term is replaced by the following expression which results from an application of L'Hopital's rule and the boundary condition (3.28)

\[
\lim_{\theta \to 0} \left( \frac{1}{\sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial \psi}{\partial \theta} \right) \right)
\]

\[
= \lim_{\theta \to 0} \left[ \frac{\partial^2 \psi}{\partial \theta^2} + \cos \theta \frac{\partial \psi}{\partial \theta} (\sin \theta)^{-1} \right]
\]

\[
= 2 \frac{\partial^2 \psi}{\partial \theta^2}
\]  

(3.32)

The boundary conditions (3.28) and (3.29) are unchanged by the
introduction of the $s$ coordinate. The inner boundary condition (3.23) becomes

$$\frac{\partial \psi}{\partial s} + \frac{\psi}{2} \bigg|_{s=0} = 0$$  \hspace{1cm} (3.33)

and the outer boundary condition (3.14) is merely

$$\psi \bigg|_{s=1} = 1$$  \hspace{1cm} (3.34)

To solve the boundary value problem numerically, the well-known techniques of finite differencing are used. (The reader who is unfamiliar with this topic is referred to Appendix A and the references given therein.) The continuous domain is replaced by a uniform grid as shown in Figure 4. The points marked with a 'x' in the diagram, which comprise the set

$$\{(s_i, \theta_j) \mid s_i = i \Delta s, \ i = 0, 1, \ldots, n_s-1; \ \theta_j = j \Delta \theta, \ j = 0, 1, \ldots, n_\theta\}$$

are the points at which a difference analogue of the interior equation (3.13) will be satisfied. The points marked with a 'o'

$$\{(s_{i-1}, \theta_j) \mid j = 0, 1, \ldots, n_\theta\} \cup$$
$$\{(s_i, \theta_{i-1}) \mid i = 0, 1, \ldots, n_s-1\} \cup$$
$$\{(s_i, \theta_{n_\theta+1}) \mid i = 0, 1, \ldots, n_s-1\}$$
Figure 4
Uniform Grid for Discretization of Boundary Value Problem
which lie outside of the continuous domain are introduced so that centred difference approximations may be employed for the boundary conditions as well as for the interior equation at grid points lying on the actual boundary of the domain.

Using expressions (3.31) and (3.32), and replacing all derivatives by second order centred differences, the discrete version of (3.13) is

\[
\frac{(1 - s_i)}{a^2 \Delta S^2} \left[ \Psi_{i+1, j} - 2 \Psi_{i, j} + \Psi_{i-1, j} \right]
\]

\[
+ \frac{(1 - s_i)}{a^2 \Delta \Theta^2} \left[ C_j^+ (\Psi_{i, j+1} - \Psi_{i, j}) - C_j^- (\Psi_{i, j} - \Psi_{i, j-1}) \right]
\]

\[
+ \frac{M - \sin \Theta_{i+1/2}}{\sin \Theta_j} \Psi_{i, j} = 0 \quad \text{;} \quad i = 0, 1, \ldots, n_\theta - 1
\]

\[
\text{where } C_j^+ = \begin{cases} 
\frac{\sin \Theta_{i+1/2}}{\sin \Theta_j} & j = 1, 2, \ldots, n_\theta - 1 \\
2 & j = 0
\end{cases}
\]

The boundary equations (3.33), (3.28), and (3.29) are discretized in the same fashion

\[
\frac{\Psi_{i, j} - \Psi_{i, j}}{2 \Delta S} + \frac{\Psi_{i, j} - \Psi_{i, j-1}}{2 \Delta \Theta} = 0 \quad \text{;} \quad j = 0, \ldots, n_\theta - 1
\]

\[
\frac{\Psi_{i, j} - \Psi_{i, j}}{2 \Delta S} = 0 
\]

\[
\frac{\Psi_{i, n_\theta} - \Psi_{i, j}}{2 \Delta \Theta} = 0 \quad \text{;} \quad i = 0, \ldots, n_\theta - 1
\]
Finally, (3.34) simply becomes

$$\Psi_{n_g, j} = 1 \quad ; \quad j = 0, \ldots, n_g$$  \hspace{1cm} (3.39)

These last five expressions define a system of non-linear algebraic equations in the unknowns $\Psi_{i,j}$, which when solved, provide approximations to the values of the true solution $\Psi$ at the discrete grid points. There is no theorem regarding the uniqueness of a solution to such a non-linear system. Existence is demonstrated by actually solving the system (approximately) numerically. Some of the more well-known techniques for solving non-linear systems of difference equations are described in section A.4 of Appendix A. The next chapter details the relatively new method which was used to solve the above problems.

A quantity of particular physical interest which may be calculated once $\Psi$ has been (approximately) determined is the total energy of the asymptotically flat spacetime [29] defined by

$$E = - \frac{1}{2\pi} \int_{\infty}^{\Sigma} \vec{D} \cdot \Psi \ d^2S_i$$  \hspace{1cm} (3.40)

where the integration is performed over a 2-sphere at spacelike infinity. Applying Gauss's theorem in the region $r>a$, the above becomes

$$E = - \frac{1}{2\pi} \left[ \int_{r>a} \nabla^2 \Psi \ d\Sigma + \int_{r=a} \vec{D} \cdot \Psi \ d^2S_i \right]$$
Using equations (3.13) and (3.23) and the fact that \( \Psi \) has no \( \phi \) dependence, this becomes

\[
E = \frac{1}{16\pi} \int_{r_1}^{r_2} \frac{\hat{K}_{ij} \hat{K}^{ij}}{\Psi^2} + \frac{a}{2} \int_0^\pi \Psi \sin \theta \, d\theta 
\]  

(3.41)

In practice, both of these integrals must be evaluated by numerical means and as discussed in Chapter 5, do not, in general, yield identical numerical results.

One final result of Bowen and York's analysis was very useful in testing and comparing the numerical procedures used in this thesis. They considered the following "model" \( H \), similar in form to (3.26) but having no angular dependence

\[
H_{\text{Model}} = \frac{6P^2}{r_4^2} \left( 1 - \frac{a^2}{r_4^2} \right)^2
\]  

(3.42)

They were then able to produce the following exact solution of the boundary value problem given by (3.13), (3.14) and (3.23)

\[
\Psi_{\text{Model}} = \left( 1 + \frac{2E}{r} + \frac{6a^2}{r_4^2} + \frac{2a^2E}{r^3} + \frac{a^4}{r_4^4} \right)^{1/4}
\]  

(3.43)

where the total energy \( E \), determined by (3.40) or (3.41) is

\[
E = \left( P^2 + 4a^2 \right)^{1/2}
\]  

(3.44)
This chapter describes a general numerical technique known as the multi-grid method, which may be used to solve boundary value problems such as the one formulated in the previous chapter. The method can also be applied to other continuous problems such as functional integrals or minimization problems, but these aspects will not be discussed here. Section 4.1 introduces the basic processes involved in any multi-grid algorithm. A description of the particular algorithm which was implemented for the purpose of solving the problem described in Chapter 3 follows in sections 4.2 - 4.7. (A listing of the actual program is contained in Appendix B, along with details of the implementation.) The majority of the material of this chapter is originally due to Brandt and various collaborators [4], [5], [6], [8] and the presentation closely parallels that of the above references.

4.1 Basic multi-grid processes

Consider the following general two-dimensional boundary value problem
\[ L \{ u(x) \} = f(x) \quad x \in \Omega \subset \mathbb{R}^2 \] (4.1a)
where $L$ and $B$ are differential operators corresponding to the interior and boundary equations for $u(x)$. Assume that this problem has been discretized in some prescribed fashion using finite difference techniques (see Appendix A) resulting in the discrete system

$$L^h u^h = f^h \quad ; \quad x \in \Omega^h$$  \hspace{1cm} (4.2a)

$$B^h u^h = g^h \quad ; \quad x \in \partial \Omega^h$$  \hspace{1cm} (4.2b)

The goal is to solve this discrete system for the function $u^h$ which is an approximation of $u$. For the sake of simplicity, it will be assumed here that the finite domain $\Omega$ is a rectangular grid, $G^h$, having a uniform spacing, $h$, in both directions. In addition, because the multi-grid method considers the differenced interior and boundary equations as separate systems to be solved using the same technique, only the solution of the interior equations will be described explicitly.

Suppose the system (4.2) is to be solved iteratively, which will always be the case if $L^h$ (or $B^h$) is non-linear. Then a good initial estimate of $u^h$ is highly desirable. One rather obvious way of generating such an estimate is to solve the problem

$$L^H u^H = f^H \quad \text{on} \quad G^H$$  \hspace{1cm} (4.3)

where $G^H$ is a coarser grid, having, for example, a mesh
spacing $H=2h$. This problem will involve fewer equations than (4.2) and therefore should require less work to solve. Once $u^H$ has been determined, then

$$\tilde{u}^h := \Pi^h u^H \quad (4.4)$$

may be used as the initial estimate of $u^h$. (The operator $:= \Pi^h$ means "is assigned the value"). Here $\Pi^h$ is an operator which interpolates a fine grid function from a coarse grid function. The logical extension of this idea is to solve discretized versions of (4.1) on an entire sequence of successively finer grids. While this technique is a part of the multi-grid solution process, it is commonly used in conjunction with many other schemes for the solution of difference equations. However, in the multi-grid method, the coarse grids play a second important role which will now be described.

Assume for the time being that $L^h$ is linear and that an estimate $\tilde{u}^h$ of the fine-grid unknown has been determined. Then the residual vector, $r^h$, of the system is given by

$$r^h = L^h \tilde{u}^h - f^h \quad (4.5)$$

and (4.2a) is solved by finding the correction, $v^h$, which satisfies

$$L^h v^h = -r^h \quad (4.6a)$$

$$u^h = \tilde{u}^h + v^h \quad (4.6b)$$

Of course, nothing significant has been accomplished in rewriting the problem in this form - (4.6a) is just as
difficult to solve as (4.2a). However, if there was some way of representing (4.6a) on a coarser grid $G^h$, then an estimate of $v^h$ could be obtained in a fashion analogous to the determination of $\tilde{u}^h$ from the solution of the coarse grid system (4.3). In general, (4.6a) can not be represented accurately on a coarser grid since $r^h$ may be highly oscillatory from point to point on $G^h$; that is, $r^h$ may have high frequency components that could not be reproduced on a coarser grid. In fact, this is typically the case when the fine grid solution estimate is generated from interpolation of a coarse grid solution. Therefore, before (4.6a) can be represented on $G^h$, the residual must be smoothed in some fashion, so as to remove the high frequency components. Fortunately, for most problems, this smoothing may be accomplished using well-known relaxation techniques. (See Appendix A, section A.3.1) A simple example best illustrates this.

Consider the following one-dimensional boundary value problem

$$\frac{\partial^2 u}{\partial x^2} = f(x); \quad x \in (0, 1)$$

(4.7a)

$$u(0) = u(1) = 0$$

(4.7b)

Discretize this problem by introducing the uniform grid $G^h$

$$G^h = \{ x_i \mid x_i = ih; \quad i = 0, 1, \ldots, N \}$$

(4.8)

$$hN = 1$$

Denoting $u(x_i)$ by $u_i$, the second derivative is replaced by the
second-order centred difference approximation yielding the set of difference equations

\[ \frac{-u_{i+1} + 2u_i - u_{i-1}}{h^2} = f_i; \quad i = 1, \ldots, N-1 \]  

(4.9)

\[ u_0 = u_N = 0 \]

which may be written in the form

\[ L^h u^h = f^h \]  

(4.10)

Here \( L^h \) is an \( N-1 \times N-1 \) tridiagonal matrix; \( u^h \) and \( f^h \) are \((N-1)\)-component vectors.

Consider the solution of the above system by the following iterative scheme. Let \( u^{(k)} \) be the approximation to \( u^h \) at the \( k \)-th iteration. Then

\[ u^{(k+1)} = u^{(k)} - \omega D^{-1} (L^h u^{(k)} - f^h) = u^{(k)} - \omega D^{-1} r^{(k)} \]  

(4.11)

Here, \( D \) is the main diagonal of \( L \), so

\[ D^{-1} = \frac{h^2}{2} \mathbb{I} \]  

(4.12)

\( \omega \) is a parameter which will be chosen from the interval \([0,1]\).

This method is called a damped Jacobi iteration. If \( \omega = 1 \), for example, then this iteration involves making a sweep through the points of \( G^h \), altering the value of each grid-function value so that the local difference equation is satisfied using neighboring function values from the previous iteration. Now examine the effect of this iteration on the residual vector.
\[ r^{(k+1)} = L^h u^{(k+1)} - f^h \]
\[ = L^h (u^{(k)} - \omega D^{-1} r^{(k)}) - f^h \]
\[ = (L^h u^{(k)} - f^h) - \omega L^h D^{-1} r^{(k)} \]
\[ = (I - \omega L^h D^{-1}) r^{(k)} \]
\[ = \hat{G} r^{(k)} \]  \hspace{1cm} (4.13)

with
\[ \hat{G} = I - \omega L^h D^{-1} \]  \hspace{1cm} (4.14)

Clearly
\[ r^{(k)} = \hat{G}^k r^{(0)} \]  \hspace{1cm} (4.15)

where \( \hat{G}^k \) is the \( k \)-th power of the matrix \( \hat{G} \). Now, \( \hat{G} \) possesses a complete set of orthonormal eigenvectors \( \hat{\varphi}_m, m=1,\ldots,N-1 \) with corresponding eigenvalues \( \hat{\lambda}_m \). Thus \( r^{(0)} \) may be expressed as
\[ r^{(0)} = \sum_{m=1}^{N-1} c_m \hat{\varphi}_m \]  \hspace{1cm} (4.16)

where the \( c_m \) are coefficients. It follows from (4.15) that
\[ r^{(k)} = \sum_{m=1}^{N-1} c_m (\hat{\lambda}_m)^k \hat{\varphi}_m \]  \hspace{1cm} (4.17)

From (4.12) and (4.13), it can be seen that the eigenvectors of \( \hat{G} \) are the same as those of \( h^2 L^h \), and that
\[ \hat{\lambda}_m = 1 - \frac{\omega}{2} \lambda_m \]  \hspace{1cm} (4.18)

where \( \lambda_m \) are the eigenvalues of \( h^2 L^h \). A straightforward
calculation shows that the eigenvectors and eigenvalues of $h^2Lh$ are

$$\phi_m = (\sin (\pi mh), \sin (2\pi mh), \ldots, \sin ((N-1)\pi mh)) \quad (4.19a)$$

$$\lambda_m = 4 \sin^2 \left( \frac{m\pi h}{2} \right) \quad (4.19b)$$

$m = 1, 2, \ldots, N-1$

Taking $\omega = 1/2$, this gives

$$\hat{\phi}_m = (\sin (\pi mh), \sin (2\pi mh), \ldots, \sin ((N-1)\pi mh)) \quad (4.20a)$$

$$\hat{\lambda}_m = \cos^2 \left( \frac{m\pi h}{2} \right) \quad (4.20b)$$

Now consider the eigenvalue corresponding to the lowest frequency eigenvector of $\hat{G}$

$$\hat{\lambda}_1 = \cos^2 \left( \frac{\pi h}{2} \right) = 1 - \frac{\pi^2 h^2}{4} + \mathcal{O}(h^4) \quad (4.21)$$

$$= 1 - \mathcal{O}(h^2)$$

From (4.17), it can be seen that the corresponding low frequency component of the residual will be damped very slowly, for small $h$, by this iteration, resulting in the slow asymptotic convergence rate characteristic of relaxation methods. However, this is not the case for the high frequency components of the residual vector, which will be defined as those components which can not be represented on a coarser grid $G^H$ with $H = 2h$. These components correspond to eigenvectors having wavelengths less than $4h$, that is with $ mh \leq \frac{1}{4}$. The corresponding eigenvalues satisfy

$$\hat{\lambda}_m \leq \cos^2 \left( \frac{\pi}{4} \right) = 0.5 \quad (4.22)$$
Thus, the high frequency components of the residual are damped very efficiently - a few relaxation sweeps will virtually eliminate them. In addition, each iteration reduces high frequency components by an amount which is independent of the mesh spacing, that is, the smoothing rate is independent of $h$. In a similar fashion it can be shown that the error vector

$$e^{(k)} = u^h - u^{(k)}$$  

is also smoothed by this relaxation process.

The smoothing of the residual and error vectors is a characteristic property of relaxation methods. However, the smoothing rate of a given method depends on the nature of the finite difference problem to which it is applied, and the choice of an appropriate scheme for a particular problem is, in general, a non-trivial task. This topic is discussed extensively in references [4] and [8], where both theoretical and numerical results for various types of relaxation are presented for a fairly broad class of problems.

Assume that a relaxation scheme with satisfactory smoothing properties has been determined for the current problem (4.2a). Then after a few relaxation sweeps on $G^h$, the high frequency components of $r^h$ will essentially be liquidated. At this point the system (4.6a) may be represented on the coarser grid $G^H$, since the desired correction $v^h$ will also be a smooth function. Thus, on $G^H$, the following problem is solved

$$L^H v^H = -I^H r^h$$  

(4.24)
Here, $I^H$ is a restriction operator which produces a coarse grid function from a fine grid function. Usually, the points of $G^H$ will be a subset of those of $G^h$, and $I^H$ involves a straightforward transferral of the appropriate function values. Once (4.24) has been solved, the approximation to the fine grid unknown is updated as follows.

$$\tilde{u}^h := \tilde{u}^h + I^H u^H$$

Here $I^H$ is another interpolation operator which, in practice, usually performs linear interpolation. Again, this interpolation process may introduce high frequency components in the residual but these may be effectively eliminated by a few more relaxation sweeps on the fine grid. The process of using a coarse grid to compute an approximation to $v^h$ is called a coarse grid correction. This same technique may be used to solve the coarse grid system (4.24). That is, relaxation sweeps are performed over $G^H$, updating the approximation $\tilde{v}^h$ of $v^H$ until the corresponding residual

$$r^H = I^H r^h - L^H \tilde{v}^H$$

is smoothed. At this point, an even coarser grid may be employed to compute an approximation to the defect $v^h - \tilde{v}^h$. The process continues using coarser and coarser grids until eventually, on the coarsest grid, a problem results which may be solved very inexpensively without the aid of another grid. Once this problem has been solved, a descent towards the finest grid is begun, using a series of interpolations of the various computed coarse grid corrections, each followed by a.
few more relaxation sweeps to remove residual components introduced by the interpolations. This entire process is called a coarse grid correction cycle. At the end of such a cycle, all components of \( r^h \) will essentially have been reduced by the same factor, and if the initial approximation of \( u^h \) was good, the fine grid problem may be solved to within the desired tolerance. If an even better approximation of \( u^h \) is desired, another coarse grid correction cycle may be performed.

Even though attention has been restricted to the case of linear difference equations, the preceding description illustrates the key features common to any multi-grid algorithm. There are three major ideas involved: 1) a sequence of grids with geometrically decreasing mesh sizes is employed. On each successive grid, the finite difference equivalent of (4.1a) is solved to produce an initial estimate for the unknown function on the next finer grid. 2) in the process of solving any system on any particular grid, relaxation sweeps are applied solely for the purpose of smoothing the residual of the system. (An exception is made for the coarsest grid where relaxation sweeps may be employed to actually solve the system.) 3) once the residual of a given system is sufficiently smooth, the problem of computing the necessary correction to the grid function is transferred to a coarser grid.

Quite a number of important details, such as how to determine when a residual is "sufficiently smooth", or when a given system has been "solved", have been neglected in the
description of the multi-grid method thus far. Rather than completing the description of a multi-grid algorithm for a linear problem, attention is now directed to the more general case of non-linear problems.

4.2 The full approximation storage scheme

As before, let the continuous domain of the boundary value problem be approximated by a sequence of grids denoted by $G^k$, $k=0,1,...,m$ where $G^0$ is the coarsest grid and $G^m$ is the finest grid. Again assume that each grid is uniform with mesh spacing $h_k$ which satisfies $h_k = 2h_{k+1}$, so that every other "line" of $G^{k+1}$ is a "line" of $G^k$. It should be noted that this particular choice for successive mesh spacings seems to be optimal for most problems which have been solved by the multi-grid method so far [8].

The first part of the multi-grid algorithm described in the previous section remains intact for the non-linear case. That is, for each $k$, $k=0,1,...,m$, the problem

\[ L^k \phi^k = f^k \quad (4.27) \]

is (approximately) solved, and the solution is interpolated to provide an initial estimate for the function on the next grid

\[ \tilde{\phi}^{k+1} = \Pi^k \phi^k \quad (4.28) \]

where, as in (4.4), $\Pi^k$ is a coarse-to-fine interpolation
operator. In practice, \( H^k \) usually performs polynomial interpolation. Brandt suggests general rules for determining an appropriate order of interpolation, the aim being to exploit any known "smoothness properties" of the actual solution \( u \) to as great an extent as possible. In particular, if \( L \) is a second order differential operator, and the difference system is second order in truncation error, then it is suggested that cubic interpolation be used.

To solve any of the systems (4.27), with the exception of \( k=0 \), the techniques of smoothing by relaxation and coarse grid corrections are again employed. Because the systems are non-linear, a good non-linear relaxation scheme must be found. (A few examples of such methods are described in Section A.4 of Appendix A.) Again, the choice of an appropriate relaxation scheme is problem-dependent. If the multi-grid algorithm is to perform efficiently, the relaxation method must result in a relatively high smoothing rate which is independent of \( h \), and must be fairly inexpensive to perform. In particular, the work required to complete a relaxation sweep on \( G^k \) should be \( O(n_k) \) where \( n_k \) is the number of points of \( G^k \). The reader is again referred to reference [4] for a more detailed discussion of this important topic.

Assume that a good relaxation method has been found, and that the problem on \( G^{k-1} \) has been solved. Then, on \( G^k \), relaxation sweeps are performed until the residual \( r^k \) of the system has been smoothed. In practice, this smoothing is most conveniently monitored by calculating, at each sweep, the norm of the dynamic residual vector \( \hat{r}^k \), which is the vector of
defects which are normally computed in the course of the relaxation iteration. The quantity

$$\hat{\mu} = \frac{\| \hat{r}^k \|}{\| \hat{r}^k \|}$$

(4.29)

where \( \hat{r}^k \) is the dynamic residual vector of the previous iteration and \( \| \cdot \| \) is some discrete norm, will remain fairly small (typically ~ 0.5) as long as \( r^k \) still contains high frequency components which are being efficiently damped by the relaxations. Once \( r^k \) is smooth, convergence slows, and \( \hat{\mu} \) quickly approaches a value which, for small \( h_k \), is very close to unity. Thus \( r^k \) is assumed to be smooth when \( \hat{\mu} \) exceeds some value \( \eta \) which is normally supplied as an adjustable parameter to the multi-grid algorithm. (See Appendix A of Reference [4] for more details.)

The implementation of coarse grid corrections described in section 4.1 must be modified since expressions (4.6a) and (4.6b) are not valid in the non-linear case. However, given an approximation \( \tilde{u}^k \) of \( u^k \), the problem is still to determine the correction \( v^k \) such that

$$L^k(\tilde{u}^k + v^k) = f^k$$

(4.30)

Subtract the quantity \( L^k \tilde{u}^k \) from both sides of the above to get

$$L^k(\tilde{u}^k + v^k) - L^k \tilde{u}^k = f^k - L^k \tilde{u}^k = -r^k$$

(4.31)

Now, when \( r^k \) has been smoothed, this problem may be transferred to \( G^{k-1} \). That is, on the coarse grid, the
following problem is posed

\[ L^{k-1} u^{k-1} - L^{k-1} I^{k-1} \tilde{u}^{k-1} = - I^{k-1} R^{k} \]

\[ L^{k-1} u^{k-1} = L^{k-1} I^{k-1} \alpha^{k} - I^{k-1} r^{k} \]  \hspace{1cm} (4.32)

where \( I^{k-1} \) is a fine-to-coarse restriction operator (as in (4.24)). A full explanation of this expression will be delayed until the discussion of local truncation error. For the time being, it is simply stated that the solution \( u^{k-1} \) of the above is not the same as the solution of the original \( G^{k-1} \) problem

\[ L^{k-1} u^{k-1} = f^{k-1} \]

Rather, it is the coarse grid equivalent of the function which (nearly) satisfies the fine grid equations. When an approximate solution, \( \tilde{u}^{k-1} \) of (4.32) has been determined, the quantity

\[ \tilde{u}^{k-1} = I^{k-1} \tilde{u}^{k} \]  \hspace{1cm} (4.33)

is the coarse grid approximation to the desired correction \( v^{k} \). Thus, the fine grid function is updated as follows

\[ \tilde{u}^{k} = \tilde{u}^{k} + I^{k-1} (\tilde{u}^{k-1} - I^{k-1} \tilde{u}^{k}) \]  \hspace{1cm} (4.34)

where \( I^{k-1} \) is another interpolation operator (as in (4.25)). Note that, in general \( I^{k-1} I^{k-1} \) is not equal to the identity operator, so that this is not equivalent to

\[ \tilde{u}^{k} = I^{k-1} \tilde{u}^{k-1} \]
The former expression is to be preferred since high frequency information about \( u \) which exists prior to the coarse grid correction is retained. Again, after the coarse grid correction has been completed, a few more relaxation sweeps over \( G^k \) are made to remove the rapid fluctuations in \( r^k \) which are introduced by interpolation of the correction. At this point, the \( G^k \) problem may be satisfactorily solved. If not, another coarse grid correction is initiated. As should be expected, the same technique is used to solve (4.32). A few relaxation sweeps on \( G^{k-1} \) are followed by a coarse grid correction on \( G^{k-1} \), and so on. Thus in, a complete coarse grid correction cycle, for the solution of the \( G^k \) problem, the following systems of equations are approximately solved

\[
\begin{align*}
\mathbf{L}^j \mathbf{u}^j &= \mathbf{F}^j, & j &= k, k-1, \ldots, 0 \\
\mathbf{F}^j &= \mathbf{f}^j, & j &= k \\
\mathbf{F}^j &= \mathbf{L}^j \mathbf{I}^{j+1}_j \mathbf{u}^{j+1} - \mathbf{I}^{j+1}_j \mathbf{r}^{j+1} \\
&= \mathbf{L}^j \mathbf{I}^{j+1}_j \mathbf{u}^{j+1} + \mathbf{I}^{j+1}_j (\mathbf{F}^{j+1}_j - \mathbf{L}^{j+1}_j \mathbf{u}^{j+1}) , & j \neq k
\end{align*}
\]

(4.35)

The above equations define what Brandt calls the full approximation storage (FAS) scheme, since at any stage of the solution process, the grid function \( u^j \), for any \( j \), is an approximation on \( G^j \) of the unknown on the finest grid.

To more fully understand the operation of the FAS algorithm, it is useful to introduce the concept of truncation error. Given the differential operator \( L \) of (4.1a) and a corresponding difference operator \( L^k \), the local truncation error, \( \tau^k \), defined on \( G \), is given by

\[
\tau^k = L^k \mathbf{I}^k \mathbf{u} - \mathbf{I}^k L \mathbf{u}
\]

(4.36)
where \( u \) is the exact solution of (4.1) and \( I^K \) is an operator which restricts a function of the continuous domain to the grid \( G \). The value of \( \tau^K \) at any point of \( G \) provides a measure of how well the difference operator approximates the differential operator at that point. Note that (4.36) may be rewritten as

\[
L^K I^K u = f^K + \tau^K \tag{4.37}
\]

and, if it were possible to approximate \( \tau^K \), then the solution \( u^K \) of

\[
L^K u^K = f^K + \tau^K \tag{4.38}
\]

would be a better approximation of \( u \) than the solution of

\[
L^K u^K = f^K \tag{4.39}
\]

In a similar fashion, another type of truncation error may be introduced. Define

\[
\tau_{k+1}^K = L^K I^{k+1} u^{k+1} - I^{k+1}_K L^{k+1} u^{k+1} \tag{4.40}
\]

where \( u^{k+1} \) satisfies

\[
L^{k+1} u^{k+1} = f^{k+1} \tag{4.41}
\]

\( \tau_{k+1}^K \) is called the relative local truncation error of \( L^K \) with respect to \( L^{k+1} \), and is simply the quantity which must be added to the right hand of (4.39) so that the solution, \( u^K \), of the resulting system coincides with \( I^K_{k+1} u^{k+1} \). Now, although
\( \tau_{K+1} \) cannot be determined exactly without solving (4.41) exactly, if an approximation \( \tilde{u}^{K+1} \) of \( u^{K+1} \) is known, then

\[
\tau_{K+1} = L^K I_{K+1} \tilde{u}^{K+1} - I_{K+1} L^{K+1} \tilde{u}^{K+1}
\]  

(4.42)

approximates \( \tau_{K+1} \). The coarse grid correction equation (4.32) may now be rewritten with the aid of (4.31) and the last expression

\[
L^{K-1} u^{K-1} = L^{K-1} I_{K} \tilde{u}^{K} - I_{K} L^{K} \tilde{u}^{K} + I_{K} f^{K}
\]

\[
= \tilde{\tau}_{K-1}^{K} + f^{K-1}
\]  

(4.43)

By the above reasoning, the solution, \( u^{K-1} \), of this system will coincide (nearly) with \( I_{K-1} u^{K} \), as was previously claimed.

The relative truncation error estimates, which a multigrid procedure produces, are also useful for providing natural convergence criteria, that is, quantities used for determining when a fine grid problem should be considered solved. The rule adopted here is the following: a problem is solved when the norm of the (dynamic) residuals is roughly the same size as the norm of the local truncation error of the difference scheme. It can easily be shown [6], that if the difference equations are of order \( p \) and \( h_K = 2h_{K+1} \), then \( \tau^K \) is related to the relative local truncation error as follows

\[
\tau^K \approx \frac{1}{1 - 2^{-p}} \tau_{K+1}^{K} \approx \frac{1}{1 - 2^{-p}} \tilde{\tau}_{K+1}^{K}
\]  

(4.44)

so that for the purposes of determining when to stop solving the \( G^K \) problem, calculation of \( \| \tilde{\tau}_{K+1}^{K} \| \) will suffice. However,
\( \tilde{\zeta}_{k+1} \) is not defined in the multi-grid algorithm until the solution on \( G^{k+1} \) has begun, by which time the problem on \( G^k \) has presumably been solved. Fortunately, the following relation may also be easily derived

\[
\tilde{\zeta}_{k+1} \approx \frac{1}{4} \tilde{\zeta}_{k} \approx \frac{1}{4} \tilde{\zeta}_{k-1} \tag{4.45}
\]

The quantity \( \tilde{\zeta}_k \) may be calculated as soon as a coarse grid correction is initiated on \( G^k \). Thus on \( G^k \), the solution process stops when

\[
\| \tilde{\zeta}_k \| < \alpha \| \tilde{\zeta}_{k-1} \| \tag{4.46}
\]

where \( \alpha \) is typically 0.25, but in general is a tunable parameter of the algorithm.

4.3 Solution on the coarsest grid

Any problem encountered on the coarsest grid \( G^0 \), be it the first of the systems approximating the continuous problem, or a coarse grid correction of one of the subsequent problems, must, of course, be solved without the aid of coarse grid corrections. This usually presents no problem, since \( G^0 \) will typically have so few points, that even if many relaxation sweeps must be performed to attain convergence, the net work expended will be essentially negligible in comparison to that involved in making even a single sweep over the finest grid, \( G^M \). A more crucial point that must be noted is that, as mentioned in Appendix A, a system of equations may be solved by relaxation only if the system (or more correctly for the
non-linear case - the linearized system) is positive definite (or negative definite). If this is not the case, that is, if for any of the \( m+1 \) problems, the system has \( l \) eigenvalues less (greater) than \( 0 \), then the corresponding \( l \) lowest frequency components of the residual may actually be magnified by relaxation sweeps. Now, on any grid but \( G^o \), these low frequency components are to be annihilated by coarse grid corrections, but since the multi-grid method dictates that relaxation also be employed in the correction process, these components may continue to be magnified. The only place where they may be effectively damped is on \( G^o \) and only then by employing a direct method for the solution of the coarsest grid systems. (See Appendix A, section A.3.1) Again, this will not usually degrade the efficiency of the multi-grid algorithm significantly, provided the number of points \( n_o \) in \( G^o \) is small enough so that the \( O(n_o^2) \) operations typically required to solve the sparse linearized systems directly, represents only a fraction of the operations needed to relax the finest grid. On the other hand, \( G^o \) must contain enough points to be able to effectively represent all of the components of any of the \( m+1 \) problems which may be magnified by relaxation sweeps.

4.4 Treatment of boundary conditions

As mentioned in section 4.1, the multi-grid method treats the interior and boundary difference equations independently. This means that the systems

\[
B^k u^k = g^k , \quad k = 0, 1, \ldots, m
\]
are solved in the same manner as the interior equations, using the techniques of smoothing and coarse grid correction. Of course, in the case of Dirichlet boundary conditions, there is nothing to "solve"; the boundary values of $u$ are simply determined from restriction of the given boundary values of $u$. The following discussion applies to the case of Neumann or Robbins (mixed) conditions.

As discussed in section A.2 of Appendix A, it is a common practice in the solution of difference systems to combine the boundary equations and interior equations to form a single system. Alternatively, if an iterative technique is employed, and the boundary equations are linear, then the appropriate grid function values might be updated after each pass over the interior equations, so as to satisfy the boundary equations exactly. However, such a process, which reduces the boundary residuals to zero, tends to introduce large residuals in the neighboring interior difference equations. This may result in a serious deterioration of the smoothing rate of the interior relaxation scheme which will reduce the overall convergence rate. Noting that, just as in the interior case, the boundary residuals need only be smoothed, Brandt suggests that a more appropriate procedure is to sweep over the grid points associated with the boundary equations (see Figure 20), adjusting each function value so that the resulting boundary residual is the average of the residuals of the two neighboring equations. Brandt also claims that making two such passes per interior relaxation sweep usually ensures that the boundary residuals are smoothed as efficiently as the interior
residuals. When treated in this fashion, the coarse grid correction process may then be applied to the boundary systems using formulae exactly analogous to (4.35).

4.5 Implementation of the FAS algorithm

A pseudo-code form of a FAS multi-grid algorithm suitable for the solution of a non-definite problem is shown in Figure 5. This algorithm, with a few modifications which are described in section 4.7 and Appendix B, was used to solve the boundary value problem formulated in the previous chapter. The main routine - PROCEDURE MULTI_GRID - is responsible for successively solving the discrete problems \( (L^K u^K = f^K ; B^K u^K = g^K) \) on grids \( G^K \), \( k=0,1,...,m \). The number of grids to be used is supplied as a parameter, along with the coarse grid spacing \( h_0 \). Also given are three control parameters \( \alpha, \delta, \) and \( \eta \), an initial estimate, \( u^0 \), of the unknown on the coarsest grid, and a convergence criterion \( \varepsilon^0 \) for the initial coarse grid problem. The parameters \( \alpha \) and \( \eta \) have been discussed previously, \( \delta \) is used to provide convergence criteria for coarse grid corrections. In practice all three of these parameters may be used to tune the algorithm. This topic is discussed extensively in Appendix A of [4].

The second routine - PROCEDURE SOLVE_ON_GRID - is a function of two parameters: \( k \) indicates which of the \( m+1 \) problems is being solved, \( \ell \) is the grid on which the problem is currently being solved. The flow of the routine is straightforward. Any system on the coarsest grid is solved by
PROCEDURE MULTI_GRID(m, h₀, α, δ, η, u₀, ε₀)
PERFORM SOLVE_ON_GRID(0,0)
FOR k = 1 .. m
hₖ := ½ hₖ₋₁
uₖ := Iₓ uₖ₋₁
Fₖ := fₖ
Gₖ := gₖ
εₖ := εₖ₋₁
PERFORM SOLVE_ON_GRID(k,k)
END FOR
END PROCEDURE

PROCEDURE SOLVE_ON_GRID(k, λ)
IF λ = 0 THEN
solve L^λ u^λ = F^λ; B^λ u^λ = G^λ directly
ELSE
∥r∥ := RELAX(λ)
WHILE ∥r∥ > ε^λ DO
IF ∥r∥ / ∥r∥ > γ THEN
{convergence is slow - start coarse grid correction}
ζ⁻¹ := L⁻¹ I_x⁻¹ u⁻¹ - 1_l⁻¹ L u⁻¹
ξ⁻¹ := B⁻¹ I_x⁻¹ u⁻¹ - 1_l⁻¹ B u⁻¹
IF λ = k THEN
ε⁻¹ := δ ∥r∥
END IF
u⁻¹ := I_x⁻¹ u⁻¹
F⁻¹ := F⁻¹ + ζ⁻¹
G⁻¹ := G⁻¹ + ξ⁻¹
ε⁻¹ := δ ∥r∥
{invoke routine recursively to perform coarse grid correction}
PERFORM SOLVE_ON_GRID(k, λ⁻¹)
{interpolate correction}
u⁻¹ := u⁻¹ + I_x⁻¹ (u⁻¹ - I_x⁻¹ u)
END IF
∥r∥ := ∥r∥
END WHILE
END IF
END PROCEDURE

Figure 5
FAS algorithm
direct means. In any other case, relaxation sweeps are performed over \( G^\ell \) until the problem is solved or slow convergence is detected. The routine \textsc{relax} performs the relaxation of both the interior and boundary systems and returns the norm of the dynamic residuals. If convergence is slow, a coarse grid correction is started. The right hand sides, \( F^{\ell-1} \) and \( G^{\ell-1} \), of the coarse grid systems are updated using the coarse grid correction equations. It should be noted that the quantities \( r_1^{\ell-1} \) and \( r_2^{\ell-1} \) represent relative truncation error estimates of the original interior and boundary difference equations only in the case \( \ell = k \). In this case, \( \| r_1^{\ell-1} \| \) is used to calculate the convergence criterion \( \varepsilon^\ell \) as described in section 4.2. The convergence parameter for the coarse grid system, \( \varepsilon^{\ell-1} \), is updated using \( \delta \) and then \textsc{solve_on_grid} is invoked recursively to actually perform the coarse grid correction. When the correction has been completed, \( u^\ell \) is updated and relaxation sweeps are resumed. When the norm of the dynamic residuals is less than the tolerance \( \varepsilon^\ell \), the routine terminates.

4.6 Work estimates

When properly tuned, the multi-grid method provides a very efficient means of solving quite general boundary value problems. In this section, a rough analysis of the work required to solve a typical problem using a multi-grid algorithm is made.

The majority of the work expended in a multi-grid
solution is used to perform relaxation sweeps over the various grids. Brandt claims that numerical experiments have shown that other processes, such as interpolations or injections, account for no more than 30% of the total computational work. A single relaxation sweep over any grid $G^K$ should take $O(n^K)$ operations where $n^K$ is the number of points of $G^K$. Furthermore, the coefficient of this order estimate will be the same for all grids if, as is usual, the difference equations are of the same form on all grids. Denote by $w^K$, the number of operations needed to perform a single $G^K$ relaxation sweep. If the problem is two-dimensional and $h_{k-1} = 2h_k$, then $n_{k-1} \approx \frac{1}{4} n_k$ and $w^{k-1} \approx \frac{1}{4} w^k$. Now, assuming that the relaxation method used has a smoothing rate which is independent of the mesh size, then on any grid, an essentially constant number of sweeps, $p$, must be made before a coarse grid correction is started. Following a coarse grid correction, another $q$ sweeps will, in general, be required to smooth out components introduced by interpolation of the correction. At this point, the problem may be solved; if not, another coarse grid correction is made. Each such correction should reduce the magnitude of the residual vector by a roughly constant factor. Suppose that, at most, $c$ coarse grid corrections are required to solve any problem. Then the total relaxation work, $W^K$, used to solve on $G^K$ is

$$W^K \approx (p + cq) w^K + c W^{k-1}$$  \hspace{1cm} (4.47)$$

where $W^{k-1}$ is the relaxation work needed to solve the $G^{k-1}$.
problem. Clearly,

$$W^K \leq (P + \sigma q)W^K \left(1 + \frac{1}{4}\sigma + \frac{1}{16}\sigma^2 + \cdots + \left(\frac{\sigma}{4}\right)^{k-1}\right) + \sigma^K W^0$$ (4.48)

where $W^0$ is the work required to compute a direct solution on $G^0$ which will be assumed to be constant. Now if $\sigma < 4$, then

$$1 + \frac{1}{4}\sigma + \frac{1}{16}\sigma^2 + \cdots + \left(\frac{\sigma}{4}\right)^{k-1} < \frac{1}{1 - \sigma/4}$$

so

$$W^K \leq \frac{\omega^K (P + \sigma q)}{1 - \sigma/4} + \sigma^K W^0$$

Since $w^K \approx cn_K$ and $\sigma^k < \frac{1}{4}\sigma^k \leq n_K/n_0$, this becomes

$$W^K \leq n_K \left[\frac{c(P + \sigma q)}{1 - \sigma/4} + \frac{W^0}{n_0}\right]$$ (4.49)

with the conclusion, that the relaxation work, and therefore the total work necessary to solve the problem on any grid is proportional to the number of points in the grid. This behavior has been verified in numerical experiments involving a wide variety of boundary value problems, and is probably the most compelling reason for employing the multi-grid method. None of the methods commonly used for the solution of finite difference systems provide $O(n_K)$ performance for general problems. For example, as discussed in sections A.3.2 and A.4 of Appendix A, the commonly used successive-over-relaxation (SOR) technique requires at least $O(N_0^3)$ operations in general.
This result assumes that the number of grid points in either direction is roughly the same, so that $N_k \approx \sqrt{N_k}$. The work estimate for the multi-grid method is then $O(N^2)$.

4.7 Adaptive discretization

In the description of the multi-grid algorithm so far, the sequence of grids employed have all approximated the entire continuous domain of the boundary value problem. In addition, because the coarsest grid spacing and number of grids to be used are supplied as parameters to the algorithm, it is implicitly assumed that the user has some idea of how fine a mesh is required to provide a satisfactory solution of the problem. In practice, this assumption may be far from realistic, since it presumes a priori information about the unknown function. Moreover, the grid spacing needed to yield a solution of uniform accuracy will in general vary from place to place in the solution domain. If each grid used covers the entire domain, much work may essentially be wasted by using fine grids in regions where they are not really necessary.

One attractive concept, which can be incorporated into a multi-grid scheme with relative ease, is to allow the ultimate discretization of the continuous domain to be determined in the course of the solution process. Such a technique is called adaptive discretization and, when properly implemented, can enhance the effectiveness and efficiency of the algorithm considerably. A general discussion of adaptive techniques in the context of the multi-grid method would require another chapter. The current discussion will be limited to a brief
description of the simple technique used in the solution of the problem formulated in the previous chapter. The interested reader is referred to [4], [5], or [6] for considerably more detailed discussions.

The basic goal of adaptive discretization is to provide a solution of uniformly high accuracy with a minimum of wasted work. To achieve this, it is first necessary to have some means of estimating the accuracy of a computed solution at a given place in the problem domain. In addition, generating such an estimate should not require so much work that the computational advantage the adaptive technique is supposed to provide is lost. In general, estimating the accuracy of an approximate solution directly is difficult, if not impossible, since the exact solution is usually unknown. However, in the multi-grid method, the local accuracy of the difference scheme can be estimated using the local truncation error estimates that the algorithm routinely generates. For properly constructed difference schemes, as the local truncation error becomes very small, so does the local error in the computed solution. Thus, an ability to control the level of truncation error also provides a way of controlling the accuracy of the solution. One might also expect that a uniform level of truncation error in all parts of the domain might also imply a relatively uniformly accurate solution.

These ideas are incorporated into the multi-grid program which is listed in Appendix B. The program accepts a parameter, $T_c$, which represents an upper bound on the desired level of truncation error. When the submitted problem has been
solved on some grid $G^k$, a local truncation error estimate $\tilde{\varepsilon}^k$ on $G^k$ is generated. This estimate is then examined to determine if there is some region covered by $G^k$ where $\tilde{\varepsilon}^k$ consistently exceeds $\varepsilon_c$. If such a region is discovered, then a new, finer grid $G^{k+1}$ is introduced, only in that region. As a result, the sequence of grids used do not, in general, all extend over the entire domain. Finer and finer grids are introduced, only where it is felt they are required, until the estimated local truncation error is less than $\varepsilon_c$ everywhere in the domain. The modifications to the FAS algorithm are straightforward and are discussed in Appendix B. The implementation of this technique was also aided by knowledge of the results Piran had previously calculated. Most notably, it was known that the solutions sought were all nearly spherically symmetric. Therefore, the finest level of discretization required in the angular direction was supplied as a parameter to the program, and adaptive discretization was performed only in the radial direction. In addition, it was determined experimentally that the truncation error estimates on any grid tended to decrease monotonically with increasing radial distance which allowed further simplifications of the adaptation process.

In conclusion, the multi-grid method can produce solutions to discretized boundary value problems in an efficient manner, while also providing an estimate of the accuracy of the difference scheme used. The major drawback to the method would seem to be in its implementation. Coding a multi-grid algorithm is not a trivial task and this may
explain why the method has not yet enjoyed widespread use.
CHAPTER 5

Numerical Experiments and Results

This chapter presents the results of some numerical tests performed to evaluate the multi-grid algorithm as well as the results of the application of the method to the boundary value problem of Chapter 3.

5.1 Comparison of multi-grid method with other methods

The first series of numerical tests were designed to compare the performance of the multi-grid algorithm with two SOR-Newton methods and a Newton-direct scheme which used a nested dissection ordering. The reader is referred to Appendix A for descriptions of the latter three methods. It was originally intended to carry out the tests on the boundary value problem described in chapter 3 using the "model" H given by equation (3.42) so that the solutions obtained from the various methods could be checked against the exact solution (3.43). However, the inner boundary condition (3.33) makes the system of difference equations non-definite with the result that the relaxation methods fail to converge to a solution. Therefore, a modified test problem, which was definite, was used so that the efficiency of the relaxation methods, which are similar to those which have been previously used in the
solution of elliptic problems in numerical relativity [15], [41], could be evaluated with respect to the multi-grid method.

For each method, the modified test problem, with $P=2$ and $a=1$, was solved on 4 uniform grids having mesh spacings

\[ \Delta s = 2^{-n} \]
\[ \Delta \theta = \frac{\pi}{2} \cdot 2^{-n} \]

(5.1)

(In addition, the problem was solved for $n=7$ using the multi-grid method.) Each routine was supplied with the same initial estimates for the unknown function on each grid, and the same convergence criteria. These were determined in the following way. For each $n$, the multi-grid program was used to solve the actual, non-definite test problem to level $n-1$. The interpolated estimate for level $n$ was then stored to be used as the initial estimate for the other three methods. At this point, the flow of the multi-grid routine was altered in a fashion which effectively replaced the Neumann and Robbins boundary conditions of the problem with Dirichlet conditions, making the problem definite. The multi-grid program proceeded to solve the level $n$ problem until the norm of the dynamic residuals was less than the norm of the estimated relative truncation error. When this had been accomplished, the norm of the actual residual was computed to serve as the convergence criterion for the other three methods.

The other three routines took the estimate generated by the multi-grid program and solved the modified problem (Dirichlet boundary conditions) until the norm of the true
residual was less than the convergence criterion output by the multi-grid method. (The norm of the true residual was monitored, because the dynamic residuals generated for a given function estimate vary from method to method).

All of the routines used were written in "standard" FORTRAN and compiled using the IBM H optimizing compiler. The tests were performed on an Amdahl 470 V/8 CPU operating under the MTS time-sharing operating system. Two quantities were measured to characterize each run—execution time and memory storage. The execution times were determined using a system timing routine and were reproducible to within a percent or so even under varying degrees of system load. The memory storage figures were hand calculated from the dimensions of the major arrays used for each method and do not include object code requirements. All calculations were performed using IBM double precision (8 byte) variables and arithmetic.

Table I shows the test results for the multi-grid method. The total relaxation work performed on all levels is listed in terms of the equivalent number of sweeps, \( n_R \), on the finest level. Not included in this total is the work expended in solving the coarsest grid (5x5) systems directly which represents a significant fraction of the total execution time for the coarser systems (9x9, 17x17), but an essentially negligible amount for the finest system. The constancy of \( n_R \), as well as the timing figures themselves, show that the work required for solution is basically proportional to the number of grid points. Also note that the storage requirements are also essentially linear in the number of grid points.
Actually, for any but the smallest problem, about 40% of this storage is unnecessary, as it was used to maintain several additional functions per grid which could have been calculated when needed, rather than being stored, at the expense of a 5-10% increase in execution time.

Table II shows the results for the point-SOR method. Near-optimal values of the relaxation parameter, $\omega$, were determined experimentally for each grid and are listed in the table. Also given are the number of relaxation sweeps, $n_\omega$, needed for convergence which would appear to increase at least like the number of points on a side of the grid. On the coarser grids, this method outperforms the multi-grid algorithm due to the latter's use of a direct method on the coarsest grid. On the finer grids however, the multi-grid method is clearly superior. Note that the storage requirements
Table II

Point-SOR results for modified test problem

The results for the line-SOR method are given in Table III. The line relaxation method used here, in the special case

Table III

Line-SOR results for modified test problem

of $\omega=1$, is identical to the relaxation scheme used in the multi-grid algorithm. The results are quite similar to those
of the point-SOR method, except that on the 65 x 65 grid, divergence occurred for any $\omega > 1$. The convergence with $\omega = 1$ on this grid was very slow and the solution process was aborted after 100 iterations. From the asymptotic convergence rate, it was estimated that at least 1500 more iterations would be required to solve the system.

Finally, the results of the Newton-nested dissection method are shown in Table IV. On all grids, because the initial estimates were good, only one Newton iteration was required for convergence. This meant that only a single LU decomposition, which accounts for the majority of the work in this method, had to be performed for each problem. The execution time performance of this method is about the same as the relaxation methods on the 2 finest grids, although the memory requirements are much greater. It must be emphasized that this method, unlike the relaxation methods, could be used

<table>
<thead>
<tr>
<th>Grid</th>
<th>Time (msec)</th>
<th>Storage (kbytes)</th>
<th>$| r |$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9x9</td>
<td>100</td>
<td>18</td>
<td>2.1(-5)</td>
</tr>
<tr>
<td>17x17</td>
<td>370</td>
<td>48</td>
<td>3.4(-7)</td>
</tr>
<tr>
<td>33x33</td>
<td>1170</td>
<td>240</td>
<td>7.6(-8)</td>
</tr>
<tr>
<td>65x65</td>
<td>17000</td>
<td>1400</td>
<td>1.0(-8)</td>
</tr>
</tbody>
</table>

Table IV
Newton-ND results for modified test problem
to solve a non-definite problem such as the original test problem. However, the multi-grid method shows a clear advantage over the direct method, both in execution time and storage requirements.

5.2 Testing of the adaptive multi-grid algorithm

The second set of numerical tests performed was intended to test the accuracy of the multi-grid algorithm operating in an adaptive mode as described in the previous chapter. For these tests, the proper boundary conditions were used in conjunction with the model H of (3.42) so that the calculated solution could be compared with the exact solution (3.43). The input to the algorithm consisted of the momentum $P$, and the parameter $\tau_c$ which controlled the discretization in the $s$ direction. Because the test problem has a spherically symmetric solution, a maximum of two levels of discretization in the $\theta$ direction was employed. When the algorithm completed the solution at a given level, a new, finer grid was introduced only in the region where it was estimated that the relative local truncation error exceeded $\tau_c$. If this region was null, the algorithm terminated.

The total energies of the computed solutions were also calculated numerically for comparison with the exact energies given by equation (3.44). Routines for evaluating both expressions (3.40) and (3.41) numerically were coded. Romberg integration [13] was employed in both cases. To test the integration routines, the exact values of $\gamma$ and $H_{\text{MODEL}}$ were
supplied on a 65 x 65 grid for various values of P. Table V shows the results. Clearly, the volume integral formula for

<table>
<thead>
<tr>
<th>P</th>
<th>E exact</th>
<th>E surf.</th>
<th>Error (%)</th>
<th>E vol.</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.8284</td>
<td>2.8242</td>
<td>0.15</td>
<td>2.8284</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>6.3246</td>
<td>6.2559</td>
<td>1.1</td>
<td>6.3246</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>10.198</td>
<td>9.9436</td>
<td>2.5</td>
<td>10.199</td>
<td>0.01</td>
</tr>
<tr>
<td>14</td>
<td>14.142</td>
<td>13.558</td>
<td>4.1</td>
<td>14.152</td>
<td>0.06</td>
</tr>
<tr>
<td>18</td>
<td>18.111</td>
<td>17.051</td>
<td>5.8</td>
<td>18.143</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table V
Testing of integration routines

the energy gives superior numerical results. (The poorer performance of the surface integral form is primarily due to the inaccuracy involved in computing $\nabla \cdot \psi$ numerically). As a result, the volume integral form was used to calculate the energies associated with all of the solutions which are described in this, and subsequent sections.

The results of a series of test runs with fixed momentum ($P=4, a=1$) and varying values of $\tau_c$ are listed in Table VI. The first column lists the supplied value of $\tau_c$. The next 6 columns show the extent of the various grids used in the course of each solution. The number at the head of each of these columns is the maximum number of grid points available in the s-direction at each level, and the numbers beneath
Table VI

Effect of varying truncation error parameter

indicate the position of the outer boundary of the corresponding grid for each run. The execution times listed are for solution of the entire problem, which, in each case, began with a solution on a 5x5 grid. In the final two columns, the relative percentage errors in the computed solutions, calculated as

$$\frac{\| \psi_{i,j}^{\text{exact}} - \psi_{i,j}^{\text{computed}} \|}{\| \psi_{i,j}^{\text{exact}} \|} \times 100 \quad (5.2)$$

using both $L_1$ and $L_\infty$ norms are listed. These last two columns show how the error in the computed solution uniformly decreases as the convergence criteria becomes more stringent, even though the entire domain is not being discretized in a uniform fashion.

Table VII shows the results of a series of runs in which $\tau_c$ was held constant at $10^{-5}$ and the momentum was varied. The
format of this table is the same as Table VI. The execution
time needed increases with the momentum, but remains
essentially linear in the total number of grid points used at
the various finest levels. The error in the computed solution,
as a function of momentum, is quite constant, varying from

<table>
<thead>
<tr>
<th>P</th>
<th>Extent of grids</th>
<th>Time (msec)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9 17 25 45 --</td>
<td>2000</td>
<td>.058 .052</td>
</tr>
<tr>
<td>4</td>
<td>9 17 33 53 73</td>
<td>3225</td>
<td>.041 .036</td>
</tr>
<tr>
<td>6</td>
<td>9 17 33 57 93</td>
<td>3700</td>
<td>.081 .062</td>
</tr>
<tr>
<td>8</td>
<td>9 17 33 57 101</td>
<td>3750</td>
<td>.132 .090</td>
</tr>
<tr>
<td>10</td>
<td>9 17 33 57 105</td>
<td>3800</td>
<td>.196 .121</td>
</tr>
<tr>
<td>12</td>
<td>9 17 33 65 109</td>
<td>4200</td>
<td>.081 .046</td>
</tr>
<tr>
<td>14</td>
<td>9 17 33 65 113</td>
<td>4300</td>
<td>.106 .053</td>
</tr>
<tr>
<td>16</td>
<td>9 17 33 65 113</td>
<td>4500</td>
<td>.133 .062</td>
</tr>
<tr>
<td>18</td>
<td>9 17 33 65 117</td>
<td>4400</td>
<td>.165 .070</td>
</tr>
<tr>
<td>20</td>
<td>9 17 33 65 117</td>
<td>4850</td>
<td>.194 .078</td>
</tr>
</tbody>
</table>

Table VII
Multi-grid test results for varying momentum

.04% to .12% in the $L_1$ norm. Again, this demonstrates the
effectiveness of the adaptive procedure in producing solutions
of uniform accuracy with a minimum of wasted work. Figure 6
shows the energies computed and a plot of the exact values

$$E = \left( P^2 + 4 \alpha^2 \right)^{1/2} = \left( P^2 + 4 \right)^{1/2}$$

(5.3)
At P=20, the deviation of the calculated energy from the exact value is about 1%. The energy estimate tends to be less accurate than the solution estimate for larger values of P, since as P increases, the volume integral term of (3.41) contributes more to the total energy and the exponentiation of $\gamma$ tends to magnify the error in $\gamma$.

Finally, to provide a visual representation of some of the features of the multi-grid algorithm discussed in the previous chapter, the effects of relaxation sweeps and coarse grid corrections on the residuals of a typical (P=4) problem are shown in Figures 7 and 8. The solid line in Figure 7 connects the values of the residuals $r_{i,j}$, $i=0,...,32; j=9$ as computed on a 33x17 grid immediately following interpolation of the solution computed on a 17x17 grid. The highly oscillatory nature of the residuals is clearly visible. However, after 5 relaxation sweeps, the residuals have evidently been smoothed. At this point, a coarse grid correction is performed. The solid line of Figure 8 shows the residuals as computed immediately after the interpolation of the correction. Again, the residuals show strong fluctuations on the scale of the grid spacing. Following the application of 3 more relaxation sweeps, the residuals are once again smoothed and another coarse grid correction can now be performed.

For all of the above test problems, as well as the problems to be described in the next two sections, 3 coarse grid correction cycles were generally required at each level to solve within the level of truncation error. It seemed that
Figure 6
Energy vs. Momentum for Model Problem
Figure 7
Pre-CGC Smoothing of Residuals

- = AFTER INITIAL INTERPOLATION
x = AFTER 5 RELAXATION SWEEPS
Figure 8

Post-CGC Smoothing of Residuals
the non-definiteness of the problem tended to slow the overall convergence rate, because when the problems were modified by imposing Dirichlet conditions as described in the previous section, only a single correction cycle was usually required. It is possible that a different relaxation scheme, or a reformulation of the boundary conditions might improve the performance, but these possibilities have not yet been investigated.

5.3 Numerical results for boosted black holes

This section presents the results obtained from application of the adaptive multi-grid algorithm to the discretized boundary value problem (3.35)-(3.38), with \( H_{ij} \) determined by one of \( H^r_B \) of equation (3.26). For both \( H^r_B \) and \( H^r_\theta \), the problem was solved for a series of linear momentum values chosen to correspond with those used by Piran in his work on the problem. In all cases the radius \( a \) was chosen to be 1 and \( r_c \) for all of the runs was \( 10^{-6} \). Table VIII lists the calculated total energies associated with each momentum value using \( H^r_\theta \) and \( H^r_B \). Note that there is very little difference in the calculated energies for a given momentum between the two cases. Figure 9 shows a plot of the data for \( H^r_\theta \). Qualitatively, the plot is very similar to the one in Figure 6. Judging from the results of the previous section, the calculated energies are probably accurate to within a percent or two. Both sets of results agree with those obtained by Piran to within two percent. (Piran does not state which of
Figure 9

Energy vs. Momentum for Boosted Black Holes
$H^+_H$ or $H^-_H$ was used in the calculation of his results). Figure

<table>
<thead>
<tr>
<th>$P$</th>
<th>$E(H^+_B)$</th>
<th>$E(H^-_B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.35</td>
<td>2.33</td>
</tr>
<tr>
<td>2.5</td>
<td>3.59</td>
<td>3.55</td>
</tr>
<tr>
<td>5.0</td>
<td>6.15</td>
<td>6.09</td>
</tr>
<tr>
<td>7.5</td>
<td>8.87</td>
<td>8.81</td>
</tr>
<tr>
<td>10.0</td>
<td>11.6</td>
<td>11.5</td>
</tr>
<tr>
<td>12.5</td>
<td>14.4</td>
<td>14.3</td>
</tr>
<tr>
<td>15.0</td>
<td>17.2</td>
<td>17.1</td>
</tr>
<tr>
<td>17.5</td>
<td>20.0</td>
<td>20.0</td>
</tr>
</tbody>
</table>

Table VIII
Total energy vs. momentum for boosted holes

10 shows the radial behavior of $\gamma$ (using $s$ coordinates) for three values of $P$, using $H^+_B$. Portions of the same data are plotted in Figure 11 against the standard radial coordinate ($r=8.0 \rightarrow s=0.875$). All of the solutions obtained were very nearly spherically symmetric. Figure 12 shows the small departure from spherical symmetry at $s=0$ of the same $\gamma$’s plotted in the previous two figures.

Piran also used the solutions he determined numerically to calculate the positions of the apparent horizons of the various boosted black holes. This information was then used to estimate the amount of gravitational radiation which might be
Figure 10

S Dependence of Conformal Factor of Boosted Holes
Figure 11

$R$ Dependence of Conformal Factor of Boosted Holes

- $\square = P=2.5$
- $\div = P=7.5$
- $\times = P=12.5$
Figure 12
Angular Dependence of Conformal Factor of Boosted Holes
present in the systems. These calculations have not yet been performed with the current data.

5.4 Numerical results for spinning black holes

Another series of multi-grid runs were performed using the squared conformal extrinsic curvature given by equation (3.27), again with $a=1$ and $\zeta_c=10^{-5}$. A plot of the calculated total energy versus angular momentum appears in Figure 13. The range of $J$ values used was again chosen to facilitate comparison with Piran's results. Radial cross sections of the conformal factor for 3 values of $J$ are displayed using $s$ and $r$ coordinates in Figures 14 and 15 respectively. The computed solutions for large values of $J$ show significant departure from spherical symmetry near the inner boundary of the solution domain as can be seen in Figure 16. A plot of $\gamma$, for $J=1000$, in the region $1 \leq r \leq 8$, $0 \leq \theta < \frac{\pi}{2}$ appears in Figure 17. The asymmetry near $r=1$ is clearly visible.

Bowen and York have shown that the spacetimes generated from evolution of the initial data set $[\psi^A f^i_j, \psi^{-10} K^i_j(\bar{J})]$ will not be Kerr spacetimes. However, they hypothesize that the spacetimes will be asymptotically Kerr; that is, it is expected that the rotating black holes will emit gravitational radiation and eventually "settle down" to a Kerr configuration. Piran and York used this reasoning to estimate the amount of gravitational radiation which could be present in the new spacetimes. The Christodolou formula [12] relates the total energy $M$ of the (neutral) Kerr spacetime to the
Figure 13
Energy vs. Momentum for Spinning Black Holes
Figure 14
S Dependence of Conformal Factor of Spinning Holes
Figure 15

R Dependence of Conformal Factor of Spinning Holes
Figure 16
Angular Dependence of Conformal Factor of Spinning Holes
Figure 17
Conformal Factor of Spinning Black Hole (J=1000)
irreducible mass, $M$, and the angular momentum $J$, of the hole

$$M^2 = M_{ir}^2 + \frac{J^2}{4M_{ir}^2} \quad (5.4)$$

The irreducible mass is related to the area, $A$, of the apparent horizon of the Kerr hole as follows

$$M_{ir} = \left( \frac{A}{16\pi} \right)^{1/2} \quad (5.5)$$

For the case of one of the new spinning black holes, Bowen and York argue that the apparent horizon coincides with the $r=a$ surface. The mass, $M_{AH}$, associated with the apparent horizon is defined as

$$M_{AH} = \left( \frac{A_{AH}}{16\pi} \right)^{1/2} \quad (5.6)$$

The area of the apparent horizon, $A_{AH}$, can be easily calculated numerically (once $\gamma$ has been determined) from the following expression

$$A_{AH} = \int_0^{2\pi} \int_0^\pi a^2 \gamma^4(a, \theta, \phi) \sin \theta \, d\theta \, d\phi \quad (5.7)$$

which in the current case, is simply

$$A_{AH} = 2\pi \int_0^\pi \gamma^4(a, \theta) \sin \theta \quad (5.8)$$

Making the assumptions that the black hole will not lose angular momentum as it settles to a Kerr configuration, and
that M will not decrease, York and Piran conclude that the quantity

\[ M \left( M_{AH}, J \right) = \left( M_{AH}^2 + \frac{J^2}{4 M_{AH}^2} \right)^{1/2} \]  

represents an asymptotic lower bound on the sum of the irreducible and rotational contributions to the total energy of the system. If this is the case, then the quantity

\[ \Delta E = E - M \left( M_{AH}, J \right) \]  

is an upper limit on the amount of gravitational radiation present in the system.

Following York and Piran, the values of three parameters characterizing the new rotating black holes are plotted in Figure 18. The quantities

\[ \epsilon_\chi \equiv \frac{J}{E^2} \]  

and

\[ \epsilon_\omega \equiv \frac{J}{M^2 (M_{AH}, J)} \]

are analogous to the Kerr angular momentum parameter

\[ \epsilon \equiv \frac{J}{M^2} \]

which tends to unity as the angular momentum of the Kerr hole
Figure 18
Angular Momentum and Energy Parameters for Spinning Holes
becomes very large. If the assumptions of the previous paragraph are valid, then \( \epsilon_L \) and \( \epsilon_U \) represent lower and upper bounds on the value of \( \epsilon \) for the Kerr black hole to which the new hole is asymptotic. It appears that \( \epsilon_L \) approaches an asymptotic limit of approximately \( 0.92 \); \( \epsilon_U \) also seems to approach a limiting value of about \( 0.98 \). For large values of \( J \), the computed energies are probably only accurate to 1 or 2 percent, so it is possible that this family of holes has an extreme Kerr limit. These results differ from those discussed by Piran and York. They find \( \epsilon_L \) and \( \epsilon_U \) tending to limiting values of about \( 0.33 \) and \( 0.55 \) respectively. An examination of Piran's code for the solution of the constraint equation (for the rotating holes only) suggests that the factor of \( 1/8 \) appearing in the non-linear term of equation (3.13) may have been omitted in the calculation. Because \( H_\phi \) is quadratic in \( J \), the values of \( J \) supplied to Piran's algorithm probably corresponded to actual values which were a factor \( t \) times larger. Among other things, this implies that his asymptotic limit for \( \epsilon_L \) should also be multiplied by \( \sqrt{8} \), which yields a value of about \( 0.93 \), which is in good agreement with the current result. (In fact all of the previous results for \( \epsilon_L \), \( \epsilon_U \), and \( J/M^2_{\lambda\lambda} \) agree with the current calculations to within two percent when the factor of \( \sqrt{8} \) is taken into account).

Finally, it would appear that \( \Delta E \), as given by equation (5.10) is considerably smaller for large values of \( J \) than had previously been calculated. From Table IX, it can be seen that the maximum value of \( \Delta E \) is about 3% for the three configurations of largest angular momentum. Piran and York had
obtained a figure of about 25% for \( J=1000 \). This would seem to indicate that this family of black holes may radiate less than had been previously expected. However, no firm conclusions can be made about the radiative nature of these systems until the initial data has actually been evolved.

Table IX

Various energies of spinning black holes
The results presented in the previous chapter show that the multi-grid method was quite successful in producing numerical solutions of the Hamiltonian constraint for the new families of black holes. Although the performance of the method was not optimal in the sense that three coarse grid correction cycles were generally needed to solve a problem at any given level of discretization, the amount of work expended was still linear in the number of fine grid points used. Thus, the computational advantage of using the method, as opposed to a more traditional scheme such as SOR, increased as finer grids were used. Furthermore, because of the non-definiteness of the problems, straightforward relaxation methods could not have been used to solve the desired equations. The only other method examined which was capable of producing solutions to the "real" problems - the Newton-nested dissection scheme - was also outperformed by the multi-grid method. Finally, the feasibility of using adaptive techniques in conjunction with the multi-grid method was demonstrated. It is difficult to imagine how adaptive discretization could be easily incorporated into the other three methods.

Of all the problems encountered in the course of
designing and debugging the multi-grid program used to obtain the results of Chapter 5, the non-definiteness of the difference equations and the treatment of the non-Dirichlet boundary conditions proved to be the most troublesome. The fact that the difference systems were non-definite went undetected through the initial testing of the program, where only two or three relatively coarse levels of discretization were used. In these tests, the difference equations were "solved" on the coarsest level by relaxation. Because the solution process on a given grid was terminated when the norm of the residuals was less than some convergence criterion, the fact that a few components were being magnified by the relaxation sweeps while the rest were being damped, was not obvious. It was only when relatively fine grids were finally used that it was seen that the coarse grid corrections were not effective and that overall divergence was actually occurring. It was at this point that the routines for solving the coarsest systems directly were coded and added to the program.

Although Brandt's suggestions for the smoothing of the non-Dirichlet boundary conditions were implemented in the program, it is not clear whether the treatment is completely satisfactory. Because of the introduction of "off-boundary" points to allow the use of centred difference approximations everywhere, the coarse grid correction process involved an extrapolation of the fine grid function values near the boundary to produce a coarse grid representation of the fine grid function. It is possible that the inaccuracy involved in
this extrapolation contributed to the slower overall rate of convergence of the method when the proper boundary conditions were used as compared to that obtained when Dirichlet conditions were imposed (as in the first series of numerical tests described in the previous chapter). The treatment of the boundary conditions certainly warrants further investigation. It would probably be instructive to investigate the effect of using non-centred difference equations near boundaries, so as to avoid the introduction of grid points extraneous to the solution domain.

As noted in the introductory chapter of this thesis, one of the main purposes of applying the multi-grid method to the Hamiltonian constraint for the new black hole families was to investigate its effectiveness as a possible tool for the solution of more complicated initial value problems. General initial value problems involving all three spatial dimensions, as well as most two-dimensional problems will undoubtedly require the numerical solution of all four of the initial value equations. The multi-grid method has previously been applied to non-linear systems of elliptic equations [4], [8]. The basic processes involved are the same as those for a scalar boundary value problem. Again, it has been the experience of researchers that the construction of a properly smoothing relaxation scheme for the given system of equations is the first priority for the efficient operation of the multi-grid algorithm. Moreover, it seems that there are very few general "rules" for designing good relaxation schemes for equations resulting from the discretization of an arbitrary
elliptic system.

In addition to the design of appropriate relaxation methods, other factors which would have to be dealt with in the multi-grid solution of a general initial value problem include the treatment of boundary conditions and the unboundedness of the domain of the problem in the case of asymptotically flat initial data. One of the basic premises of the multi-grid philosophy is that useful information can be extracted from a very coarse approximation of the continuous problem of interest. For very large domains, it may not always be possible to meaningfully approximate the problem on a grid with so few unknowns that the solution of the resulting system of algebraic equations may be performed with an essentially negligible amount of work. In such cases, the work expended in the multi-grid solution process may not be proportional to the number of fine grid points.

On the positive side, one might expect that the multi-grid method, combined with adaptive discretization, would allow a more accurate solution of a given initial value problem than SOR, for example, for a given amount of computational work. As stated at the end of Chapter 4, the price that must be paid for the computational efficiency of the multi-grid method is the substantial human effort required to implement it. In comparison to the development of the program listed in Appendix B, the coding of the point- and line-SOR methods was almost trivial. It must be noted, however, that the the major portion of the multi-grid program was designed so that it could be easily modified for the
solution of a different boundary value problem. Brandt makes the observation, which would appear to be fairly accurate, that most of the routines used in a well written multi-grid program for the solution of a particular boundary value problem can be used with little or no modification for other similar problems. This is little consolation, however, to a prospective "multi-griddler" who may not have access to an existing piece of software which could be easily modified for her use. "General purpose" multi-grid software has been written, but it is probable that a fair amount of effort and expertise would be required to use it effectively.

Finally, apart from the initial value equations, other elliptic equations, which could be solved using the multi-grid method, arise in some current and proposed approaches to the numerical evolution of gravitational initial data [32], [40]. In addition, there has been some success in the application of the multi-grid scheme to time dependent problems. For example, Brandt et al [7], have applied the method to hydrodynamical problems wherein some of the variables are treated implicitly in time to allow the use of a relatively large time step. The possibility of using a similar technique for gravitational evolution problems would seem to be worthy of investigation.
Bibliography


22. James, M.L., G.M. Smith and J.C. Wolford. 


24. --------. "Canonical Quantization of Gravity" 
In *Relativity, Astrophysics and Cosmology.*


Cambridge: Cambridge University Press, 1979, pp. 139-159.


APPENDIX A

Basic Numerical Techniques for Elliptic PDE's

This appendix reviews some of the basic concepts and methods used in the numerical solution of elliptic boundary-value problems using finite differences. No attempt is made to describe all of the available techniques, nor is any particular method presented in detail. The material in sections A.1 - A.2 is intended to give the reader a general idea of the methods used to convert boundary-value problems, such as the one described in Chapter 3, to finite difference form. Sections A.3 - A.4 describe some of the "classical" methods of solving the systems of algebraic equations which result from such problems, with emphasis placed on the particular methods which were used in this thesis for the purpose of comparison with the multi-grid method detailed in Chapter 4.
A.1 Discretization

A general boundary-value problem in d dimensions may be written in the form

\[ L \{ u(x) \} = f(x) \quad x = (x_1, \ldots, x_d) \in \mathcal{D} \subset \mathbb{R}^d \]  \hspace{1cm} (A.1a)

\[ B \{ u(x) \} = g(x) \quad x \in \partial \mathcal{D} \]  \hspace{1cm} (A.1b)

Here L and B are linear or non-linear differential operators, f(x) and g(x) are specified functions and u(x) is the unknown function. For simplicity, assume that the domain \( \mathcal{D} \) is a rectangular region of \( \mathbb{R}^2 \) as depicted in Figure 19a. Thus

\[ \Omega = \left\{ (x,y) \mid a \leq x \leq b \ , \ c \leq y \leq d \right\} \]  \hspace{1cm} (A.2)

The first step in the discretization process is to replace the continuous domain by a discrete domain. The simplest way to do this is to introduce a grid on the continuous domain as shown in Figure 19b. In this case the grid has uniform spacings h in the x direction and k in the y direction and is therefore called a uniform grid. The finite set of grid points is the discrete domain \( \Omega^h \)

\[ \Omega^h = \left\{ (x_i, y_j) \mid a \leq x_i \leq b \ , \ c \leq y_j \leq d \right\} \]  \hspace{1cm} (A.3)
Figure 19
Discretization of Domain of Boundary Value Problem
where
\[ x_i = a + ih; \quad i = 0, 1, \ldots, N; \quad N = h^{-1}(b - a) \]
\[ y_j = c + jk; \quad j = 0, 1, \ldots, M; \quad M = k^{-1}(d - c) \]  

(A.4)

The boundary of the discrete domain, \( \partial \Omega^h \), is made up of those grid points which lie on the boundary of the continuous domain. A restriction operator \( I^h \) is now introduced which operates on an arbitrary function defined on \( \Omega \), yielding the finite set of function values corresponding to the elements of \( \Omega^h \). The second step of the discretization process involves replacing the continuous differential operators \( L \) and \( B \) with discrete operators \( L^h \) and \( B^h \) such that the solution \( u^h \) of the system
\[
L^h u^h = I^h f \quad \text{on} \quad \Omega^h \quad \text{(A.5a)}
\]
\[
B^h u^h = I^h g \quad \text{on} \quad \partial \Omega^h \quad \text{(A.5b)}
\]

is an approximation to \( I^h u \). One method for constructing such discrete operators, the technique of finite differencing, is the topic of the next section.

A.2 Finite Differences

Assume that the solution \( u(x, y) \) to the above problem is sufficiently well behaved that it may be expanded in a Taylor series about any point in the domain \( \Omega \).
Specifically, consider the following expansion for the value 
\[ u(a + (i+1)h, c + jk) \approx u(x_i, y_j) \]

\[ u(x_{i+1}, y_j) = u(x_i, y_j) + h \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h^2}{2!} \frac{\partial^2 u}{\partial x^2}(x_i, y_j) + \ldots \quad (A.6) \]

Solving the above for \( \frac{\partial u}{\partial x}(x_i, y_j) \) yields

\[
\frac{\partial u}{\partial x}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - u(x_i, y_j)}{h} + \frac{h}{2} \frac{\partial^2 u}{\partial x^2}(x_i, y_j) + \ldots \quad (A.7)
\]

Introducing the short-hand notation,

\[ u_{i,j} \equiv u(x_i, y_j) \]

the first order forward difference approximation for \( \frac{\partial u}{\partial x} \) is

\[ u_{i+1,j} - u_{i,j} \]

The approximation is called first order since the truncation error of the expression, resulting from truncation of the Taylor series, is of order \( h \). In a similar fashion, an expansion for \( u_{i-1,j} \) may be performed

\[ u_{i-1,j} = u_{i,j} - h \frac{\partial u_{i,j}}{\partial x} + \frac{h^2}{2!} \frac{\partial^2 u_{i,j}}{\partial x^2} - \ldots \quad (A.9) \]

\[
\frac{\partial u_{i,j}}{\partial x} = \frac{u_{i,j} - u_{i-1,j}}{h} + O(h)
\]

This yields the first order backward difference formula for \( \frac{\partial u_{i,j}}{\partial x} \)

\[ u_{i,j} - u_{i-1,j} \]

\[ h \quad (A.10) \]
Another approximation may be constructed by subtracting (A.9) from (A.6)

\[ u_{i+1,j} - 2u_{i,j} + u_{i-1,j} = 2h \frac{\partial^2 u}{\partial x^2} + \frac{h^3}{6} \frac{\partial^3 u}{\partial x^3} + \ldots \]  

(A.11)

\[ \frac{\partial u_{i,j}}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O(h^2) \]

yielding the second order central difference approximation for \( \frac{\partial u_{i,j}}{\partial x} \)

\[ \frac{u_{i+1,j} - u_{i-1,j}}{2h} \]

(A.12)

Difference approximations for higher derivatives may be obtained in a similar fashion. For example, the second order central difference approximation of \( \frac{\partial^2 u_{i,j}}{\partial x^2} \) is

\[ \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \]

(A.13)

Second order central difference formulae are probably the most commonly used approximations in finite differencing of elliptic boundary value problems. Higher order approximations, while more accurate for a given grid spacing, in general require more work numerically, both for their evaluation and for the solution of the algebraic equations resulting from their use. No elementary discussion of difference techniques for elliptic equations would be complete without mention of the 5-point approximation for the Laplacian, \( \nabla^2 u \), in the special case when the grid spacing in the y direction is also \( h \)

\[ \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} \]
This, too, is a second order central difference formula. As a final example of simple differencing techniques, consider the following expression

\[ \nabla \cdot [a(x, y) \nabla u(x, y)] \]  \hfill (A.14)

where \(a(x, y)\) is a specified function. Expressions of this sort often arise in physical applications which involve some sort of macroscopic conservation law. For example, using Gauss's theorem

\[
\int_{\partial \Omega} \nabla \cdot [a(x, y) \nabla u(x, y)] \, d\mathbf{n} = \int_{\Omega} a(x, y) \nabla u(x, y) \cdot d\mathbf{n} \]  \hfill (A.15)

If (A.14) is discretized in the following manner, using second order central differences

\[
\left[ \frac{1}{h} \left[ a_{i+\frac{1}{2}, j} (u_{i+\frac{1}{2}, j} - u_{i, j}) - a_{i-\frac{1}{2}, j} (u_{i, j} - u_{i-\frac{1}{2}, j}) \right] + \frac{1}{k} \left[ a_{i, j+\frac{1}{2}} (u_{i, j+\frac{1}{2}} - u_{i, j}) - a_{i, j-\frac{1}{2}} (u_{i, j} - u_{i, j-\frac{1}{2}}) \right] \right] \]

where \(a_{i+\frac{1}{2}, j} \approx a(x_{i+\frac{1}{2}}, y)\); \(a_{i, j+\frac{1}{2}} \approx a(x_{i}, y_{j+\frac{1}{2}})\), then the solution of the difference equations will automatically satisfy a discrete version of (A.15)

\[
\text{LHS of (A.15)} \approx \sum_{i, j} \frac{1}{h^2} \left[ a_{i+\frac{1}{2}, j} (u_{i+\frac{1}{2}, j} - u_{i, j}) - a_{i-\frac{1}{2}, j} (u_{i, j} - u_{i-\frac{1}{2}, j}) \right] + \frac{1}{k^2} \left[ a_{i, j+\frac{1}{2}} (u_{i, j+\frac{1}{2}} - u_{i, j}) - a_{i, j-\frac{1}{2}} (u_{i, j} - u_{i, j-\frac{1}{2}}) \right] \approx \text{RHS of (A.15)}
\]
However, if (A.14) is rewritten as
\[ a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial^2 u}{\partial x \partial y} + b \frac{\partial^2 u}{\partial y^2} + c \frac{\partial u}{\partial x} + c \frac{\partial u}{\partial y} \]
and differenced as follows
\[ a_{i,j} \frac{1}{h^2} \left[ u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right] + \left( \frac{\partial a}{\partial x} \right)_{i,j} \frac{1}{2h} \left[ u_{i+1,j} - u_{i-1,j} \right] \]
\[ + a_{i,j} \frac{1}{k^2} \left[ u_{i,j+1} - 2u_{i,j} + u_{i,j-1} \right] + \left( \frac{\partial a}{\partial y} \right)_{i,j} \frac{1}{2k} \left[ u_{i+1,j} - u_{i-1,j} \right] \]
then the solution of the difference equations will not obey a discrete version of (A.15). Difference expressions such as (A.16) are called conservative, since the discrete unknowns obey a discrete version of a conservation law. Conservative difference schemes are discussed in depth in [25]. General finite differencing techniques are treated in detail in [1] and [19].

There are three major types of boundary conditions encountered in boundary-value problems: 1) Dirichlet conditions, where the value of the unknown function is given on the boundary, 2) Neumann conditions, where the normal derivative of the function on the boundary is specified, and 3) Robbins or mixed conditions, where some combination of the function and its first derivative on the boundary are given. Dirichlet conditions present little difficulty when formulating finite difference schemes. However, to implement the other two types of conditions, it is often useful to introduce extra lines of grid points which lie outside of the actual continuous domain, as depicted in Figure 20. This allows a centred difference approximation to the normal derivative to
Figure 20

Extra Grid Points - Discretization of Boundary Conditions
be employed.

The net result of replacing the operators $L$ and $B$ of (A.1) with finite difference approximations $L^h$ and $B^h$ is to convert the continuous differential problem into a finite system of algebraic equations. In general, there will be one such equation for each grid point in $\Omega^h$ ($\Omega^h - \partial \Omega^h$) for problems with Neumann/Robbins (Dirichlet) conditions, resulting from the discretization of the interior equation (A.1a). In addition, the discretization of differential boundary conditions will effectively supply an equation for the value of the function at any extra grid points which may have been introduced as described above. These "extra-boundary" function values will typically be referenced by one interior difference equation and one boundary equation, and it is common practice to eliminate this value from the two equations, effectively incorporating the boundary equation into the interior equation. In any case, a properly constructed difference scheme using a grid with a total of $n$ points will result in a system of $n$ equations for the $n$ values of the grid function $u$. Because a typical finite difference approximation involves only the value of the function at the point of application and a few neighboring function values, any given equation in the system will involve only a few of the unknowns. The next sections of this appendix describe some of the methods for solving such sparse systems of equations numerically.
A.3 Methods for solving linear systems

If the operators $L$ and $B$ in (A.1) are both linear, then the system of $n$ equations resulting from the discretization of the complete boundary value problem will also be linear and may be written in the form

$$A \mathbf{u} = \mathbf{b}$$  \hspace{1cm} (A.17)

where $A$ is a $n \times n$ matrix and $\mathbf{u}$ and $\mathbf{b}$ are $n$-component vectors. For the purpose of providing rough comparisons of the effectiveness of various methods for solving such linear systems, assume that the number of grid points $N$ in the $x$ direction is about the same as the number of points $M$ in the $y$ direction - that is $N \approx M \approx \sqrt{n}$. Where possible, methods will be compared using 'order of $N$' estimates of two quantities - operations required to solve the system and computer memory needed to perform the solution. Here, an operation will typically be a multiplication, division, or addition of two floating point numbers. Such arithmetic operations usually account for the majority of execution time of any of the methods to be described.

Methods for solving large, sparse linear systems fall into two general classes - direct methods and iterative methods. These classes are discussed separately in the following two sub-sections.
A.3.1 Direct methods

A direct method for solving the system (A.17) usually involves decomposing the matrix A into lower and upper triangular factors using some variant of Gaussian elimination. (The reader who is unfamiliar with Gaussian elimination is referred to any elementary text in linear algebra or numerical analysis.) (A.17) becomes

\[ A \mathbf{y} = L A U A \mathbf{y} = \mathbf{b} \]  

(A.18)

where $U_A$ and $L_A$ are $n \times n$ upper triangular and lower triangular matrices respectively. Once the decomposition has been accomplished, the solution vector is determined by solving successively the systems

\[ L_A \mathbf{y} = \mathbf{b} \]  

(A.19)

and

\[ U_A \mathbf{y} = \mathbf{y} \]  

(A.20)

The solution of such triangular systems is readily accomplished and for large values of $n$, represents only a small fraction of the work needed to perform the factorization. As a consequence, research efforts have concentrated on ways of improving the factorization process.

The major problem encountered in factoring a matrix
arising from a finite difference scheme involves the phenomenon of fill-in - the upper and lower triangular factors will in general have non-zero elements which are zero in the corresponding positions of the original matrix. Since the operations needed to perform the factorization, as well as the minimum memory storage required, depend on the number of non-zeros in $L_A$ and $U_A$, much work has been devoted to the design of algorithms which reduce fill-in.

The amount of fill suffered by the matrix $A$ in the course of the factorization process depends on the way in which the unknowns $u_i$, $i=1,...,n$ are ordered. For example, if the system to be solved results from the discretization of an elliptic problem using second-order centred differences, and the unknowns $u_i$ are numbered using a natural ordering such as the one depicted in Figure 21 for a 10 x 10 grid, then it can be shown that although there are only $O(N^2)$ non-zeros in $A$, there will be $O(N^3)$ non-zeros in each of the triangular factors $L_A$ and $U_A$, and the amount of work required to complete the factorization is $O(N^4)$. This is essentially a worst-case behaviour and several methods for reordering the unknowns to improve this performance have been developed. Of these, the most successful for the type of difference system described above is probably the nested dissection method due to George [17], [18]. Figure 22 shows a nested dissection ordering of a 10 x 10 grid. George has shown that such an ordering leads to $O(N^2\log N)$ non-zeros in the triangular factors and $O(N^3)$ operations to compute the factorization, which represents a significant improvement, for large $N$, over the natural
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 21**

Natural Ordering of a 10x10 Grid
<table>
<thead>
<tr>
<th>78</th>
<th>77</th>
<th>85</th>
<th>68</th>
<th>67</th>
<th>100</th>
<th>29</th>
<th>28</th>
<th>36</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>75</td>
<td>84</td>
<td>66</td>
<td>65</td>
<td>99</td>
<td>27</td>
<td>26</td>
<td>35</td>
<td>19</td>
</tr>
<tr>
<td>80</td>
<td>79</td>
<td>83</td>
<td>70</td>
<td>69</td>
<td>98</td>
<td>31</td>
<td>30</td>
<td>34</td>
<td>21</td>
</tr>
<tr>
<td>74</td>
<td>73</td>
<td>82</td>
<td>64</td>
<td>63</td>
<td>97</td>
<td>25</td>
<td>24</td>
<td>33</td>
<td>18</td>
</tr>
<tr>
<td>72</td>
<td>71</td>
<td>81</td>
<td>62</td>
<td>61</td>
<td>96</td>
<td>23</td>
<td>22</td>
<td>32</td>
<td>17</td>
</tr>
<tr>
<td>90</td>
<td>89</td>
<td>88</td>
<td>87</td>
<td>86</td>
<td>95</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
<td>60</td>
<td>46</td>
<td>45</td>
<td>94</td>
<td>10</td>
<td>9</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>51</td>
<td>59</td>
<td>44</td>
<td>43</td>
<td>93</td>
<td>8</td>
<td>7</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>58</td>
<td>48</td>
<td>47</td>
<td>92</td>
<td>12</td>
<td>11</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>49</td>
<td>57</td>
<td>42</td>
<td>41</td>
<td>91</td>
<td>6</td>
<td>5</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 22**

Nested Dissection Ordering of a 10x10 Grid
ordering. It has also been shown \cite{21} that these order estimates are both optimal. It should be mentioned that the ordering of the unknowns is sometimes important for another reason. Reordering the components of the vector $\mathbf{u}$ in (A.17) implies that the rows and columns of the matrix $\mathbf{A}$ must also be permuted. However, for a general matrix $\mathbf{A}$, it may be necessary to permute the rows and/or columns of $\mathbf{A}$ as the elimination procedure progresses to minimize numerical instabilities which arise if the matrix is poorly conditioned. The condition number, $\text{cond}(\mathbf{A})$, of the matrix, provides a measure of how "close" the matrix is to being numerically singular.

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$$

$$1 \leq \text{cond}(\mathbf{A}) \leq \infty$$

where $\|\cdot\|$ in the above is some matrix norm, such as

$$\|\mathbf{A}\| = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}$$

with $\|\cdot\|$ some vector norm. When $\text{cond}(\mathbf{A})$ is very large, $\mathbf{A}$ is said to be ill-conditioned. The need to interchange the rows and/or columns of $\mathbf{A}$ in the course of the solution may lead to a disruption of the ordering intended to minimize fill-in and a compromise must usually be established between the increase in memory usage and execution time due to fill-in, and the loss of accuracy due to numerical instability. Fortunately, matrices arising from the discretization of elliptic systems are usually well-conditioned so that the problem of instability does not arise and an ordering may be established
before the factorization process commences.

Another important consideration in the design of algorithms to solve a large sparse system directly is the manner in which the matrix and its factors are actually stored in the computer memory. Implicit in the above discussion of fill-in is the fact that it is not feasible to store the entire matrix A and its factors in their entirety. If $n=100$, for example, which is not unreasonable, 100 million words of computer memory would be required to store A explicitly, the vast majority of which would contain 0! Most algorithms employ sparse matrix storage techniques to minimize the number of zero elements which must be stored. Thorough discussions of many such schemes are found in [18].

A.3.2 Iterative methods

In an iterative method for solving (A.17), starting with some initial estimate of the solution $\hat{u}^{(o)}$, a series of iterates $\hat{u}^{(k)}$ is generated such that

$$\lim_{k \to \infty} \hat{u}^{(k)} = \hat{u}$$  \hspace{1cm} (A.21)

Thus, in contrast with direct methods which, neglecting the inexact nature of machine arithmetic, yield the exact solution after a finite number of steps, iterative methods require, in principle, an infinite number of operations to determine $\hat{u}$. In practice, the solution process is generally terminated after a finite number of iterations, when it is felt that the estimate
$u^{(k)}$ is "close enough" to the solution $u$. A common way to monitor the progress of an iterative scheme is to compute at each iteration, the residual vector $r^{(k)}$

$$r^{(k)} = A u^{(k)} - b \quad (A.22)$$

If the iteration converges, then

$$\lim_{k \to \infty} r^{(k)} = 0 \quad (A.23)$$

The iterative procedure may then be stopped when

$$\| r^k \| < \epsilon \quad (A.24)$$

where $\| \cdot \|$ denotes some discrete norm and $\epsilon$ is a convergence parameter which is typically given at the onset of the solution process.

Iterative methods are characterized by the manner in which the new iterate is calculated from previous iterates. In the general case

$$u^{(k+1)} = G^{(k)}(u^{(k)}, u^{(k-1)}, \ldots, u^{(0)}) + \xi^{(k)} \quad (A.25)$$

where $G^{(k)}$ is some operator which, as the superscript indicates may change from iteration to iteration, as may the vector $\xi^{(k)}$. A particular class of methods, which includes all of the methods to be described below, results when $G^{(k)}$ is linear, stationary, and operates only on the current solution
estimate, and the vector \( \tilde{z}^{(k)} \) is constant. Then

\[
\tilde{u}^{(k+1)} = G \tilde{u}^{(k)} + \tilde{c}
\]  

(A.26)

and \( G \) may be represented as a \( n \times n \) matrix and is often called the amplification matrix. The error vector \( \tilde{e}^{(k)} \) of the \( k \)th iteration is defined by

\[
\tilde{e}^{(k)} = \tilde{u}^{(k)} - \tilde{u}
\]

and it can easily be shown that

\[
\tilde{e}^{(k)} = G^k \tilde{e}^{(0)}
\]

\[
\tilde{e}^{(0)} = \tilde{u}^{(0)} - \tilde{u}
\]

where \( G^k \) is the \( k \)th power of the amplification matrix. The iterative procedure will converge if

\[
\lim_{k \to \infty} \| \tilde{e}^{(k)} \| = \lim_{k \to \infty} \| G^k \tilde{e}^{(0)} \| = 0
\]

(A.28)

A necessary and sufficient condition for the above to be satisfied is that the spectral radius, \( \rho(G) \), of the amplification matrix

\[
\rho(G) = \max_i | \lambda_i(G) |
\]

(A.29)

where \( \lambda_i \) are the eigenvalues of \( G \), be less than unity. In addition, \( \rho(G) \) determines how quickly the iterative method converges.
In general

$$\lim_{k \to \infty} \frac{\| e^{(k+1)} \|}{\| e^{(k)} \|} = \rho(G)$$  \hspace{1cm} (A.30)

and an asymptotic rate of convergence, $R$, may be defined as follows

$$R = \log_{10} \left( \frac{1}{\rho} \right)$$  \hspace{1cm} (A.31)

The reciprocal of $R$ yields the number of iterations which must be performed to reduce the error asymptotically by a factor of ten.

The study of the amplification matrix of a particular iterative method is very useful for establishing theoretical results concerning the convergence properties of the method. However, for any but the very simplest iterative schemes, the explicit form of $G$ becomes quite complicated, and writing the iterations in the form of (A.26) tends to make the methods look more complex than they actually are. An alternative approach is to describe the methods in a manner which closely parallels the way they are typically implemented as computer programs.

Let the $n$ points of the grid be numbered in some fashion - using a natural ordering, for example. Denote the value of the unknown grid function at grid position $i$ by $u_i$, and the elements of the matrix $A$ by $a_{ij}$. A particularly simple method, commonly called the Jacobi iteration, results from visiting each grid point in succession, changing the value of each
unknown so that using the required neighboring values from the previous iteration, the local difference equations are satisfied. That is

\[ u_i^{(k+1)} = - \sum_{j \neq i} a_{ij} u_j^{(k)} + b_i \] (A.32)

Note that there is no need to explicitly store the matrix \( A \), when this method is implemented on a computer. All that is required is sufficient information to evaluate the difference equations at each point. This feature is characteristic of most iterative methods and is one of the primary reasons that very large systems of difference equations have traditionally been solved by iterative, rather than direct methods. The bulk of the storage needed for the Jacobi method is used for maintaining two vectors of length \( n \) which contain at any time the grid function estimates of the current and previous iterations.

If the Jacobi method converges, then, in general, \( u_i^{(k+1)} \) will be a better estimate of \( u_i \) than \( u_i^{(k)} \). This suggests that an improvement on the Jacobi iteration might result by using newly calculated quantities whenever possible in the course of an iteration. Thus

\[ u_i^{(k+1)} = - \left( \sum_{j \neq i} a_{ij} u_j^{(k+1)} + \sum_{j \neq i} a_{ij} u_j^{(k)} \right) + b_i \] (A.33)

This is the Gauss-Seidel (GS) iteration which has the advantage of only requiring storage for a single vector of length \( n \) to maintain the estimate of the grid function.
Unfortunately, both of the above methods tend to have slow asymptotic rates of convergence. Typically, the number of iterations required to reduce the error by an order of magnitude is \(O(N^2)\). The number of calculations needed to perform a single iteration, or relaxation sweep, as it is often called, is \(O(N^2)\) if the number of operations required to evaluate each difference equation is \(O(1)\). Thus, the total work necessary to solve the system by either of the methods is \(O(N^4)\), while the memory required is \(O(N^2)\).

Historically, researchers who used the Gauss-Seidel method for solving systems of difference equations discovered that convergence could often be accelerated by modifying the iteration so that

\[
\hat{u}_i^{(k+1)} = \omega \hat{u}_i^{(k+1)} + (1 - \omega) u_i^{(k)}
\]

(A.34)

where \(\hat{u}_i^{(k+1)}\) is determined by the right hand side of (A.33), and \(\omega\) is called the relaxation parameter. If \(0 < \omega < 2\), then the above defines the well-known successive over-relaxation (SOR) iteration which incorporates the Gauss-Seidel iteration as a special case when \(\omega = 1\). The SOR method has been the subject of a great deal of research in the past and optimum values of \(\omega\) have been determined analytically for some special types of difference systems and domains. In these cases, the number of relaxation sweeps necessary for convergence may be \(O(N)\), leading to \(O(N^3)\) operations for solution of the system, again with \(O(N^2)\) memory storage. In general, however, the optimal
value of $\omega, \omega_{opt}$, which depends on the grid spacing, the shape of the domain, and the difference scheme employed, must be determined either through numerical experiment or by some sort of adaptive procedure which attempts to estimate $\omega_{opt}$ in the course of the solution process.

Because the above methods solve for one new function value at a time, they are often referred to as point-relaxation methods. Another class of relaxation methods involves the simultaneous update of a group of unknowns. For example, assume that the grid has been numbered using a natural ordering such as the one shown in Figure 21. Let $u_\alpha, \alpha = 1, \ldots, M$ be the N-component vectors of unknowns corresponding to all grid points having the y coordinate $y_\alpha$. Then a line-Gauss-Seidel (LGS) method involves solving equation (A.33) simultaneously for all $u_\alpha \in u_\alpha$, for $\alpha = 1, \ldots, M$. Thus one iteration of LGS requires the solution of M systems of N equations. Each such system, however, is usually quite easy to solve - generally requiring only $O(N)$ operations, so the work needed to perform a relaxation sweep is still $O(N^2)$. Clearly, one might also choose to use lines of unknowns having constant x coordinates, or even alternate the grouping choice from iteration to iteration. Line-SOR (LSOR) schemes may similarly be formulated. As with the point-iterative methods, theoretical and experimental results indicate that $O(N^2)$ LGS sweeps or at least $O(N)$ LSOR sweeps are required for convergence of a typical finite difference problem, so the work and storage estimates are of the same orders as the corresponding point-wise schemes. However the theory which has
been established for line-relaxation schemes, notably LSOR, covers a more general class of finite difference problems than the results for point-relaxation methods, and in practice, LSOR frequently outperforms SOR.

A final, very important point must be mentioned in connection with all of the above iterative schemes. In general, all of these relaxation methods will converge if and only if the system of difference equations is definite; that is, if

$$\mathbf{x}^T A \mathbf{x} > 0$$

or

$$\mathbf{x}^T A \mathbf{x} < 0 \quad \forall \mathbf{x} \in \mathbb{R}^n \quad \mathbf{x} \neq \mathbf{0}$$

(A.35)

For most difference schemes resulting from the discretization of a linear elliptic problem having Dirichlet or Neumann boundary conditions, $A$ will be definite and relaxation methods may be applied.

Thorough treatments of iterative techniques for large linear systems are found in [36] and [38].
A.4 Methods for solving non-linear systems

If either of the operators in \((A.1)\) is non-linear, then the discretization process gives rise to a system of non-linear algebraic equations which may be written as

\[
F_1(u_1, u_2, \ldots, u_n) = 0 \\
F_2(u_1, u_2, \ldots, u_n) = 0 \\
\vdots \\
F_n(u_1, u_2, \ldots, u_n) = 0
\]

or more succinctly

\[
F(u) = 0
\]

These equations must be solved iteratively since there is no existing algorithm for the direct solution of general non-linear systems. As in the case of linear iterative methods, the problem is usually considered solved when some convergence criterion has been met. Again, a quantity often monitored is the norm of the residual vector, \(r^{(k)}\)

\[
r^{(k)} = F(u^{(k)})
\]

The solution process may then be terminated when

\[
\| r^{(k)} \| < \epsilon
\]

where \(\epsilon\) is the convergence tolerance as before.

A considerable number of methods for the solution of non-
linear systems have been developed. Here, attention will be restricted to a few schemes which all involve a generalization to n dimensions of Newton's method for the solution of a single non-linear equation in one unknown. To review this method, if the equation to be solved is

$$ f(x) = 0 $$  \hspace{1cm} (A.39)  

and \( x^{(k)} \) is the current estimate of the solution, then the new estimate \( x^{(k+1)} \) is determined by

$$ x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \hspace{1cm} (A.40) $$

There are essentially two ways to extend this iteration to n dimensions in a straightforward manner. The first method involves linearizing the complete system of equations at each iteration. That is, given an estimate \( u^{(k)} \), the new iterate \( u^{(k+1)} \) is given by

$$ u^{(k+1)} = u^{(k)} + \delta u^{(k)} \hspace{1cm} (A.41) $$

where the vector \( \delta u^{(k)} \) satisfies the following linear system.

$$ F'\{ u^{(k)} \} \delta u^{(k)} = -F(u^{(k)}) \hspace{1cm} (A.42) $$

Here \( F' \) is the Jacobian matrix of the system having elements

$$ \left[ F'(u^{(k)}) \right] = \frac{\partial F_i}{\partial u_j} \bigg|_{u = u^{(k)}} \hspace{1cm} (A.43) $$
At each iteration, the system (A.43) must then be solved, using, for example, any of the methods described in the previous section for the solution of linear equations. One of the methods used for comparison with the multi-grid method in Chapter 5, employed the above iteration in conjunction with a nested-dissection direct method to solve the linear systems.

The work required to solve a non-linear system in this fashion depends crucially on the initial estimate. If the estimate is good enough, then convergence may be achieved with just a few Newton iterations, so the order of the work estimate may be the same as the estimate for the method used for solving the linear systems.

The second way of employing Newton's method in \( n \) dimensions is to successively apply the iteration to each individual equation, or a group of equations corresponding to a line of unknowns, in the spirit of the relaxation methods for solving linear systems. For example, the one-step point-Gauss-Seidel-Newton iteration is given by

\[
 u_i^{(k+1)} = u_i^{(k)} - \frac{F_i(u_i^{(k)})}{\frac{\partial F_i}{\partial u_i} u_i^{(k)}} \tag{A.44}
\]

The term one-step is used since only a single Newton iteration is performed before moving on to the next equation. (In certain cases, it may be desirable to perform more than one step). Similarly, the one-step point-SOR-Newton iteration is

\[
 u_i^{(k+1)} = u_i^{(k)} - \omega \frac{F_i(u_i^{(k)})}{\frac{\partial F_i}{\partial u_i} u_i^{(k)}} \tag{A.45}
\]
where, as before, $\omega$ is the relaxation parameter, whose optimal value will normally have to be determined from numerical experiments. These iterations may be modified so that an entire line of unknowns is updated simultaneously. Using the notation of Section A.3.2, a typical line-SOR-Newton iteration is given by

$$u^{(k+1)}_\alpha = u^{(k)}_\alpha + \omega \cdot \Delta u^{(k)}_\alpha; \quad k = 1, \ldots, M$$  \hspace{1cm} (A.46)

where $\Delta u^{(k)}_\alpha$ is determined by

$$F'_\alpha(u^{(k)}) \cdot \Delta u^{(k)}_\alpha = -F_\alpha(u^{(k)})$$  \hspace{1cm} (A.47)

$$[ F'_\alpha(u^{(k)}) ]_{ij} = \frac{\partial F_i}{\partial u_j} \bigg|_{u = u^{(k)}} \quad \forall F_i \in F_\alpha \; ; \; u_j \in \mathcal{U}_\alpha$$  \hspace{1cm} (A.48)

Again, solution of the above linear systems is usually readily accomplished. Both point-SOR-Newton and line-SOR-Newton methods were coded for comparison with the multi-grid method.

The amount of work necessary to determine a solution by non-linear relaxation methods is difficult to estimate in general. If the initial guess is good, then the order estimates may be the same as the corresponding linear relaxation methods. Better performance is certainly not to be expected.

A very good reference for the solution of non-linear systems is due to Ortega and Rheinboldt [32].
APPENDIX B

Implementation of the Multi-grid Algorithm

B.1 Grid organization for adaptive discretization

As stated in Chapter 4, the multi-grid program which is listed in this appendix is based on the algorithm presented in pseudo-code form in Figure 5. However, the actual program was designed to accommodate more general patterns of discretization than the one used in the pseudo-code algorithm. In the algorithm of Chapter 4, the uniform grids used all covered the entire domain of the problem, and each level of discretization was associated with a unique grid. For the purpose of adaptive discretization, the program allows more than one grid to be employed at each level, and the grids used may extend over sub-domains of the problem. Figure 23a illustrates an example of the type of discretization the program was designed to handle. All grids used are required to be rectangular with uniform mesh spacings in both directions. The coarsest grid used is assumed to extend over the entire domain. All grids, with the exception of the coarsest are required to have a unique father grid which is used for the purpose of coarse grid corrections. The father's domain must contain the domains of all of its sons. The mesh spacing ratios between a father/son pair of grids must be 2:1 or 1:1, with the 2:1
Figure 23
Non-uniform Discretization and Associated Tree Structure
ratio holding for at least one of the directions. In general, a set of grids which may be used in the program can be represented by a tree structure. Figure 23b shows the tree structure associated with the grids of Figure 23a.

The modifications to the FAS algorithm for such an organization of grids are straightforward. To solve a problem at level $k$, relaxation sweeps are applied to all grids defined on level $k$ until convergence on all grids is slow. A coarse grid correction is performed by updating the systems of equations for all grids on level $k-1$ having sons on level $k$. Relaxation sweeps are then performed over all grids at level $k-1$, etc. Note that in general, some of the coarse grids will serve dual purposes in such a scheme. First they will act as correcting grids for their sons and secondly, they may be the finest grids used in some regions of the solution domain.

The data structure which was designed to implement this type of grid organization is described in the program documentation which follows the text of this appendix. Each grid is labelled by an integer and various information describing the grids is stored in single dimensioned arrays. Pointer information to facilitate access to all grids at a given level, the father of a given grid, or the sons of a given grid, is also maintained. Functions may be defined on the points of a grid or on the two sets of coordinates which define the grid. All of these functions are stored in a single array, MEMORY. Pointers to the starting locations in MEMORY for the various functions are also kept in the grid data structure. The array MEMORY is also used throughout the
program to provide temporary working storage.

It should be noted that the multi-grid runs described in Chapter 5 did not demand as general a grid organization as the program was designed to allow. However, the additional execution time which results from using the more general scheme is truly negligible in comparison to the work involved in relaxation sweeps, etc. which must be performed by any implementation of the FAS algorithm.

B.2 Summary of major multi-grid routines

The program listed below contains two high level routines which essentially implement PROCEDURE MULTI_GRID of the pseudo-code algorithm. The first of these routines is a driver which simply inputs some run parameters and then calls the second routine, MGMAIN. MGMAIN calls intermediate level routines to solve the problem on successively finer levels until some maximum level of discretization has been used, or the estimated local truncation error estimate is less than the supplied convergence criterion everywhere in the discrete domain. MGMAIN is also responsible for calling the routine ADAPT which determines where a new fine grid should be introduced.

The second pseudo-code routine PROCEDURE SOLVE_ON_GRID, is implemented using four routines: CYCLE, SOLVE, CGCST, and CGCFIN. The implementation is somewhat different than the algorithm since "standard" FORTRAN does not permit recursive routines. CYCLE attempts to solve the level \( k \) problem as
follows. The routine SOLVE is called, which in turn calls the relaxation routine, LINRLX, to apply relaxation sweeps to the level \( l = k \) difference equations until the system is solved, or slow convergence is detected. If \( l \) is the coarsest level, a routine which solves the system directly is invoked. SOLVE returns a flag which indicates whether the problem has been solved or convergence is slow. In the latter case, CYCLE calls the routine CGCST which starts the coarse grid correction process. The right hand side of the level \( l-1 \) difference equations and the convergence criteria for levels \( l-1 \) and possibly \( l \) are updated. CYCLE then calls SOLVE on level \( l-1 \). When the value of the flag that SOLVE returns indicates that a problem has been solved, and the problem is not on level \( k \), a coarse grid correction has been completed. In this case, CYCLE invokes CGCFIN which updates the fine grid function; CYCLE then calls SOLVE on level \( l+1 \). When the level \( k \) problem has been solved, CYCLE returns control to MGMAIN. As a guard against program run-away, CYCLE has a parameter which limits the number of coarse grid correction cycles performed in an attempt to solve the level \( k \) problem.

B.3 Relaxation routines

As described in Chapter 4, the determination of a good relaxation scheme for a given discretized problem is vital for the proper operation of any multi-grid algorithm. Originally, a one-step point-SOR-Newton relaxation routine (see Appendix A, section A.4) was coded for use in the author's multi-grid
program. However, this scheme did not seem to have a very high smoothing rate and resulted in slow overall convergence. A line-SOR-Newton routine, LINRLX, was then implemented. This routine is capable of using lines of unknowns having constant angular or radial coordinate, and the lines may be swept in either direction—from low to high values of grid coordinate or vice versa. After some experimentation, it was found that a scheme employing lines of constant s swept from \( s=0 \) to \( s=1 \) gave the best results. This scheme was used in all of the runs described in Chapter 5. Routines to smooth the non-Dirichlet boundary equations as described in 4.4 were also coded. As Brandt had claimed, it was found that two sweeps on all boundaries per interior relaxation sweep were sufficient to maintain proper convergence of the boundary equations.

B.4 Control parameters

Once the multi-grid program had been written and debugged, some numerical experimentation was performed to determine suitable values for the three control parameters \( \alpha \), \( \eta \), and \( \delta \). The following values were finally chosen and used for all of the runs of Chapter 5: \( \alpha =0.35 \), \( \eta =0.70 \), and \( \delta =0.50 \). In addition, the program was modified to accept two additional parameters, PSWP and QSWP which governed the minimum number of relaxation sweeps which were performed (on any level) prior to and following a coarse grid correction. These parameters were introduced to prevent the initiation of a coarse grid correction (as controlled by the parameter \( \eta \) ).
before the residuals had been sufficiently smoothed. Without
the application of at least three or four sweeps prior to a
correction, the program sometimes exhibited an oscillatory
behaviour - performing repeated coarse grid corrections which
did not appear to be contributing very much to the solution of
the fine grid equations.
B.5 Listing of multi-grid program

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)
INTEGER BROTHER(50), FATHER(50), LHEAD(50), LLINK(50), SON(50), NPT(50), NX(50), NY(50), XHIB(50),
YLOB(50), YHIB(50), YLOB(50)
REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
XINC(50), YINC(50)
LOGICAL CONV(50), SLOW(50)
INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
INTEGER MEMPT
INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU
INTEGER NULL / Z80808080 /
COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHER, SON,
NX, NY, NPT, AREA, XINC, YINC,
XFSTRT, YFSTRT, GFSTRT, XHIB, YHIB,
YLOB, YHIB, ERROR, CONV, SLOW, EPSI,
NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT
COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS,
TAU, KSQR, NXFUNC, NYFUNC, NGFUNC
Pointer Information:

LHEAD(L)  >  First grid on level L
LLINK(LHEAD(L))  >  Next
LLINK(LLINK(LHEAD(L)))  >  Etc.

FATHER(G)  >  G's coarse grid correction grid
SON(G)  >  Finer grid contained by G
BROTHR(SON(G))  >  Another grid contained by G
BROTHR(BROTHR(SON(G)))  >  Etc.

Fixed Grid Information:

XINC(G)  >  Grid spacing in x direction
YINC(G)  >  Grid spacing in y direction
NX(G)  >  Number of points in x direction
NY(G)  >  Number of points in y direction
NPT(G)  >  Number of interior points in grid
AREA(G)  >  Area of grid
XLOB(G)  >  Boundary flags: .EQ. 0 -> Dirichlet conditions
YHIB(G) / .NE. 0 -> non-Dirichlet conditions
and contains number of smoothing
sweeps per interior relax. sweep

Variable Grid Information:

EPSI(G)  >  Current convergence criterion
ERROR(G)  >  Current dynamical residual norm
NRMTAU(G)  >  Norm of estimated local trunc. error
CONV(G)  >  Flags solution on grid G
SLOW(G)  >  Slow convergence on G

Function Pointers: (starting locations in MEMORY)

GFSTRT(G,I),I=1,NGFUNC -> Pts. to fcns. of grid pts.
XFSTRT(G,I),I=1,NXFUNC -> " " " x coords.
YFSTRT(G,I),I=1,NYFUNC -> " " " y coords.

Miscellaneous:

CRSLEV  >  Coarsest current level
FINLEV  >  Finest current level
NGRID  >  Current number of grids
MEMPT  >  Pointer to next available location in MEMORY
NGFUNC  >  # of fcns. defined on grid pts.
NXFUNC  >  " " " x coords.
NYFUNC  >  " " " y coords.
NULL  >  Null pointer
T O P  L E V E L  R O U T I N E S

D R I V E R  for multi-grid and numerical integration routines. Note: Must be re-compiled to change amount of memory allocation (MEMORY).

Run parameters:

P --------> Momentum
CVTAU ----> Truncation error convergence criterion
MAXLEV ----> Maximum number of levels of discretization
REFLEV ----> Level at which adaptive discretization starts
MXYLEV ----> Maximum number of levels in y-direction
LIST ----> .NE. 0 -> Enable program list / trace
           .EQ. 0 -> Disable
PHIIN ----> .EQ. 0 -> Calculate initial coarse grid est.
            .NE. 0 -> Read

Routines called:

CDATE  (System)
TIME   (System)
MGMAIN
INTGRT

MTS logical unit assignments:

2 -> (IN)  Initial estimate of coarse grid fcn
3 -> (OUT) Final calculated coarse grid fcn
4 -> (IN/OUT) Final calculated fine grid fcn
5 -> (OUT) Listing / program trace
6 -> (IN/OUT) User communication
7 -> (IN)  Run parameters

REAL*8  MEMORY(150000)
REAL*8  COMMNT(10)
REAL    DATEAR(7)
REAL*8  CVTAU, DXTOL, DYIOL, P
INTEGER  CONTRL, CPUT1, CPUT2, CPUOT, L, LIST, LUNIT.
*      MAXCYC, MAXLEV, PHIIN, PHIOUT, RESOUT, USER.
*      REFLEV, MXYLEV
DATA    DXTOL / 1.0D-4 /, DYIOL / 1.0D-4 /


DATA MAXCYC / 4 /, PHIOUT / 3 /,
* RESOUT / 4 /, LUNIT / 5 /,
* USER / 6 /, CONTRL / 7 /

INPUT RUN PARAMETERS

READ(CONTRL, 100) P, CVTAU, MAXLEV, REFLEV, MXYLEV, LIST, PHIIN
FORMAT(F10.3, E10.4, 5I4)

ECHO INPUT, DATE, TIME, ETC.

CALL CDATE(DATEAR)
WRITE(USER, 110)
110 FORMAT(' ENTER COMMENT:')
READ(USER, 120) (COMMNT(I), I = 1, 10)
120 FORMAT(10A8)
WRITE(LUNIT, 130) (DATEAR(I), I = 1, 7)
130 FORMAT(1H-, 7A4)
WRITE(LUNIT, 140) (COMMNT(I), I = 1, 10)
140 FORMAT(1H0, 10A8)
WRITE(LUNIT, 150) P, MAXLEV
150 FORMAT('OMOMENTUM:', F20.3, ' NUMBER OF LEVELS:', I3)
WRITE(LUNIT, 155) MXYLEV, REFLEV
155 FORMAT(' MXYLEV: ', 13, ' REFLEV: ', 13)
WRITE(LUNIT, 160)
160 FORMAT('— ---------MULTI-GRID BEGINS'//)

MULTI-GRID

CALL TIME(0)
CALL MGMAIN(MEMORY, 150000, P, CVTAU, MAXLEV, REFLEV, MXYLEV, MAXCYC,
* LIST, LUNIT, PHIIN, PHIOUT, RESOUT, USER)
CALL TIME(1.0, CPUT1)
WRITE(LUNIT, 170)
170 FORMAT('-- -- -- -- -- -- MULTI-GRID BEGINS'//)
WRITE(LUNIT, 180) CPUT1
180 FORMAT('MULTI-GRID CPU TIME:', T40, 18, ' MS.')

REWIND RESOUT

WRITE(LUNIT, 190)
190 FORMAT('-- -- -- -- -- -- NUMERIC',
* ' INTEGRATION ',
* ' BEGINS'//)

NUMERIC INTEGRATION ('VOLUME' INTEGRAL)

CALL TIME(0)
CALL INTGRT(MEMORY, 150000, DXTOL, DYTOL, RESOUT, LUNIT)
CALL TIME(1.0, CPUT2)
WRITE(LUNIT, 200)
200 FORMAT('-- -- -- -- -- -- NUMERIC',
* ' INTEGRATION ',
* ' COMPLETED'//)
CPUTOT = CPUT1 + CPUT2
WRITE(LUNIT, 210) CPUT2, CPUTOT
SUBROUTINE MGMAIN

MULTI-GRID MAIN ROUTINE

Main routine in implementation of FAS multi-grid algorithm. Responsible for controlling overall flow of algorithm. Discretized boundary value problem is solved on a maximum of MAXLEV levels, or until the local truncation error estimate at some level is uniformly below CVTAU. Routine is responsible for initialization of the grid data structure, generation of grid and coordinate functions, definition of all grids, and output of final calculated function.

Parameters:

MEMORY(NMEM) -> Main storage array
P ------------> Momentum
CVTAU ---------> Truncation error convergence parameter
MAXLEV -------> Max. number of levels of discretization
REFLEV -------> Level at which adaptive discretization begins
MXYLEV -------> Max. number of levels of discretization in y direction.
MAXCYC -------> Max. number of coarse grid correction cycles to be used to solve any system
LIST ---------> Listing/trace flag
LUNIT ---------> Listing unit
PHIIN ---------> .EQ. 0 -> calculate initial estimate of unknown on coarsest grid
                .NE. 0 -> read initial estimate from logical unit PHIIN
PHIOUT -------> Logical unit for output of final calculated coarse grid unknown
RESOUT -------> Logical unit for output of calculated fine grid unknown and other results
USER ---------> Logical unit for user communication

Routines called:

INIGST
NEWGRID
DEFGRD
DIRSPC
PRVPHI
GUESS
Major local variables:

Note: These variables are initialized in DATA statements - routine must be recompiled if values are to be changed.

A --------------- Inversion radius (should remain initialized to 1.0D0 since some other routines assume this value)
ALPHA ------------ Multi-grid control parameters
DELTA /
ETA /
EPSIO ---------- Initial coarse grid convergence crit.
XINCO, YINCO ---> Coarse grid (CG) x,y mesh spacings
XMNO, YMNO ------ Coords. of lower left corner of CG
XMCO, YMCO ------ Coords. of upper right corner of CG
XMNDOM, YMNDOM ---> Coords. of lower left corner of discrete domain
XMXDOM, YMXDOM ---> Coords. of upper right corner of discrete domain
INILEV --------- Label for initial level (CG)
MAXGRD --------- Maximum number of grids to be used
NXLO, NXHI, ---- Boundary flags (see description of grid data structure)
NYLO, NYHI /
PSWP ---------- Minimum number of pre-CGC sweeps
QSWP ---------- " " " post-CGC "

SUBROUTINE MGMAIN(MEMORY,NMEM,P,CVTAU,MAXLEV,REFLEV,MXYLEV,
* MAXCYC,LIST,LUNIT,PHIIN,PHIOUT,RESOUT,USER)

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)
INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
SON(50), NPT(50), NX(50), NY(50), XHIB(50),
XL0B(50), YHIB(50), YLOB(50)
REAL*8 AREA(50), EPSI(50), ERROR(50), NRMATAU(50),
XINC(50), YINC(50)
LOGICAL CONV(50), SLOW(50)
INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
INTEGER MEMPT
INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU
INTEGER NULL / Z80808080 /

COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHER, SON, NX, NY, NPT, AREA, XINC, YINC, XFSTRT, YFSTRT, GFSTRT, XLDB, XHIB, YLDB, YHIB, ERROR, CONV, SLOW, EPSI, NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT

COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS, TAU, KSQR, NXFUNC, NYFUNC, NGFUNC

INTEGER NMEM
REAL*8 MEMORY(NMEM)
REAL*8 NORM
REAL*8 A, ALPHA, DELTA, EPSILN, ETA, EXTRAP, P, XINCO, XINCL, XMNO, XMNDOM, XMXO, XMXDOM,
REAL*8 YINCO, YINCL, YMNO, YMNDOM, YMNO, YMNDOM,
REAL*8 MGNORM, AREAL, CVTAU, NWXMXD, XMX, XMXD

INTEGER SWPCNT(50)
COMMON / COMSWP / SWPCNT

DATA A / 1.0000D0 /, ALPHA / 0.3500D0 /,
* DELTA / 0.5000D0 /, EPSIO / 5.00-04 /,
* ETA / 0.7000D0 /,
* XINCO / 0.2500D0 /, YINCO / 0.2500D0 /,
* XMNO / -0.2500D0 /, XMNO / 1.0000D0 /,
* XMNDOM / 0.0000D0 /, XMNDOM / 1.0000D0 /,
* YMNO / -0.2500D0 /, YMNO / 1.2500D0 /,
* YMNDOM / 0.0000D0 /, YMNDOM / 1.0000D0 /

DATA INILEV / 1 /, MAXGRD / 10 /,
* NXLO / 2 /, NXHI / 3 /,
* NYLO / 2 /, NYHI / 2 /,
* PSWP / 3 /, QSWP / 2 /

INITIALIZE GRID STRUCTURE
MEMPT = 1
CALL INIGST(INILEV, MAXGRD)

INITIALIZE SWEEP COUNTERS
DO 10 I = 1, MAXLEV
   SWPCNT(I) = 0
CONTINUE

CALL NEWGRD(INILEV,NULL,MAXGRD,RC)
IF( RC .NE. 0 ) GO TO 130

DEFINE COARSEST GRID

GRID = NGRID
XINCL = XINCO
YINCL = YINCO

CALL DEFGRD(GRID,XMNO,XMCO,YMNO,YMCO,XMNDOM,XMCO,
* YMNDOM,YMCO,XINCL,YINCL,NXLO,NXHI,NYLO,
* NYHI,EPSIO,1.0D0)

CALL INIGRD(MEMORY(GFSTRT(GRID,KSQR)),MEMORY(GFSTRT(GRID,RHS)),
* MEMORY(XFSTRT(GRID,R)),MEMORY(XFSTRT(GRID,RP)),
* MEMORY(XFSTRT(GRID,WM)),MEMORY(XFSTRT(GRID,WM)),
* NX(GRID),NY(GRID),XMNO,YMNO,
* XINC(GRID),YINC(GRID),A,P)

ALLOCATE MEMORY REQUIRED BY DIRECT SOLVER AND SET UP AS MUCH
OF ACTUAL FINITE DIFFERENCE MATRIX AS POSSIBLE

CALL DIRSPC(MEMORY,MEMPT,NX(GRID),NY(GRID),
* MEMORY(XFSTRT(GRID,RP)),MEMORY(XFSTRT(GRID,RM)),
* MEMORY(YFSTRT(GRID,WP)),MEMORY(YFSTRT(GRID,WM)),
* XINC(GRID),YINC(GRID))

IF( PHIIN .EQ. 0 ) GO TO 20

READ IN INITIAL ESTIMATE OF PHI

CALL PRVPHI(MEMORY(GFSTRT(GRID,U)),NX(GRID),NY(GRID),
* PHIIN,RC)
IF( RC .NE. 0 ) GO TO 150
GO TO 30

CALCULATE INITIAL ESTIMATE OF PHI

CALL GUESS(MEMORY(GFSTRT(GRID,U)),MEMORY(XFSTRT(GRID,R)),
* NX(GRID),NY(GRID))

CGRID = GRID

DISABLE TAU EXTRAPOLATION

EXTRAP = 1.0D0
EPSILN = EPSIO
LEV = INILEV

L O O P . . . FOR EACH LEVEL ...

WRITE(LUNIT,200) LEV, MEMPT
200 FORMAT('****LEVEL',14, '****MEMORY',
* 'USAGE:',18, 'DOUBLE WORDS')

ATTEMPT TO SOLVE LEVEL LEV PROBLEM

CALL CYCLE(MEMORY,LEV,MAXCYC,ALPHA,DELTA,ETA,EXTRAP,
* PSWP,QSWP,LIST,LUNIT,RC)
WRITE(LUNIT,210) ERROR(GRID)
210 FORMAT('OFINAL DYNAMIC RESIDUAL NORM: ',E14.5)
IF( LEV .EQ. MAXLEV ) GO TO 76

C WAS PROBLEM SOLVED USING MAXCYC OR FEWER CGC'S
C IF( RC .NE. O ) GO TO 140
C IF SO, PROCEED TO NEXT LEVEL
C 50
LEV = LEV + 1
L1 = LEV - 1
C CALL NEWGRD(LEV,GRID,MAXGRD,RC)
IF( RC .NE. 0 ) GO TO 130
C IF( LEV .GE. REFLEV ) GO TO 60

C FULL-SIZE NEW GRID
C AREAL = 1.0DO
XINCL = 0.5DO * XINCL
IF( LEV .LE. MXYLEV ) YINCL = 0.5DO * YINCL
XMX = XMNO
XMXD = XMNO
GO TO 70

C NEW GRID DETERMINED ADAPTIVELY
C 60
L2 = LEV - 2
CALL ADAPT(NX(L2),NY(L2),
.* MEMORY(GFSTRT(L2,TAU)),
.* MEMORY(XFSTRT(L2,R)).ALPHA,
.* CVTAU,AREA(L1),AREAL,NWXMXD,
.* NOREF)
C IF( NOREF ) GO TO 74
C WRITE(LUNIT,215) LEV, NWXMXD
215 FORMAT(' OUTER BOUNDARY OF GRID ',I4,
.* XINCL = 0.5DO * XINCL
IF( LEV .LE. MXYLEV ) YINCL = 0.5DO * YINCL
XMX = NWXMXD
XMXD = NWXMXD

C 70
EPSILN = ALPHA * (EPSI(L1) * AREA(L1)) / AREAL
OGRID = GRID
GRID = NGRID
C DEFINE AND INITIALIZE NEW GRID, INTERPOLATE INITIAL ESTIMATE OF UNKNOWN FUNCTION
C CALL DEFGRD(GRID,XMNO,XMX,YMNO,YMNO,XMNDOM,
.* XMXD,YMNDOM,XMXDOM,XINCL,YINCL,
.* NXLO,NXHI,NYLO,NYHI,EPSILN,AREA)
C CALL INIGRD(MEMORY(GFSTRT(GRID,KSQR)),
.* MEMORY(GFSTRT(GRID,RHS)),
.* MEMORY(XFSTRT(GRID,R)),
.* MEMORY(XFSTRT(GRID,RP)),
.* MEMORY(XFSTRT(GRID,RP)))
CALL INTERP(MEMORY,OGRID,GFSTRT(OGRID,U),GRID,GFSTRT(GRID,U),LIST,LUNIT)

GO TO 40

NORMAL EXIT

MAXLEV = LEV - 1

LEV = MAXLEV

IF( LEV .EQ. 0 ) GO TO 160
    NXLEV1 = NX(LEV) - 1
    IF( MEMORY(XFSTRT(LEV,R)+NXLEV1) .EQ. 1.0D0 ) GO TO 90
    LEV = LEV - 1
GO TO 77

GRID = LEV

INJECT FINE GRID FUNCTIONS TO FINEST LEVEL WHICH COVERS ENTIRE DISCRETE DOMAIN

LEV = MAXLEV

IF( LEV .EQ. GRID ) GO TO 110
    LEVM1 = LEV - 1
    CALL INJECT(MEMORY(GFSTRT(LEV1,U)),MEMORY(GFSTRT(LEV,U))
    NX(LEV1),NY(LEV1),NX(LEV),NY(LEV),
    MEMORY(XFSTRT(LEV1,R)),MEMORY(YFSTRT(LEV1,W)),
    MEMORY(XFSTRT(LEV,R)),MEMORY(YFSTRT(LEV,W)),
    XINC(LEV1),YINC(LEV1),XINC(LEV),YINC(LEV),
    XLOB(LEV1),XHIB(LEV1),YLOB(LEV1),YHIB(LEV1),
    XLOB(LEV),XHIB(LEV),YLOB(LEV),YHIB(LEV),1)
    LEV = LEVM1
GO TO 100

DO 120 I = 1, MAXLEV
    WRITE(LUNIT,220) SWPCNT(I), I
    FORMAT(14,' SWEEPS ON LEVEL ',14)
CONTINUE

OUTPUT CALCULATED QUANTITIES

CALL OUTRS1(MEMORY(GFSTRT(GRID,KSQR)),MEMORY(GFSTRT(GRID,U))
    NX(GRID),NY(GRID),RESOUT)

CALL OUTRS2(MEMORY(GFSTRT(CGRID,U)),NX(CGRID),NY(CGRID),PHIOUT)

WRITE(LUNIT,230) MEMPT
    FORMAT('-MEMORY USAGE: ',16,' DOUBLE WORDS.')

RETURN
ABNORMAL EXITS

WRITE(USER,240)
FORMAT('-PROGRAM ABORTING FOLLOWING ERROR RETURN FROM S/R .
     'NEWGRD. ')
RETURN

WRITE(USER,250) MAXCYC, LEV
FORMAT('-NO CONVERGENCE AFTER ',14,' CYCLES ON LEVEL ',14/
     '<CR> TO CONTINUE, 1 TO EXIT. ')
READ(USER,260) RC
FORMA T(I1)
IF( RC .EQ. 0 ) GO TO 50
GO TO 76

WRITE(USER,270)
FORMAT('-PROGRAM ABORTING FOLLOWING ERROR RETURN FROM S/R .
     'PRVPHI. ')

WRITE(USER,280)
FORMAT('-COULD NOT FIND S=1 !!')
RETURN

END
INTERMEDIATE LEVEL ROUTINES
FOR PERFORMING COARSE GRID
CORRECTION CYCLE AND ADAPTIVE
DISCRETIZATION

SUBROUTINE CYCLE

Attempts to solve level L problem by calling SOLVE to perform
relaxation or direct solution and by performing at most
MAXCYC coarse grid correction cycles.

Parameters:

MEMORY -> Main storage array
L -------> Level of current problem
MAXCYC -> Maximum number of coarse grid correction cycles
ALPHA --> Alpha
DELTA --> Delta
ETA ---> Eta
EXTRAP --> Tau-extrapolation parameter
PSWP ----> Minimum number of Pre-CGC relaxation sweeps
QSWP ----> Minimum number of Post-CGC relaxation sweeps
LIST ---> Listing flag
UNIT ----> Listing unit
RC -------> Return code -> .EQ. 0 -> problem solved
           .EQ. 1 -> problem not solved

Routines called:

SOLVE
CGCST
CGCFIN

SUBROUTINE CYCLE(MEMORY,L,MAYCYC,ALPHA,DELTA,ETA,EXTRAP,
* PSWP,QSWP,LIST,UNIT,RC)

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)
INTEGER BROTHER(50), FATHER(50), LHEAD(50), LLINK(50),
* SON(50), NPT(50), NX(50), NY(50), XMIB(50),
* XLOB(50), YHIB(50), YLOB(50)
REAL AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
REAL*8 MEMORY(1)
REAL*8 ALPHA, DELTA, ETA, EXTRAP
INTEGER L, LEV, LIST, MAXCYC, NCYC, RC, UNIT
INTEGER MINSWP, NSWP, PSWP, QSWP
LOGICAL ALCONV

NCYC = 0
LEV = L
MINSWP = PSWP

CALL SOLVE(MEMORY,LEV,ALCONV,ETA,3,MINSWP,NSWP,LST,UNIT)
IF( LIST .GT. O ) WRITE(UNIT, 90) NSWP, LEV
FORMAT(1H ,14,' SWEEPS AT LEVEL ',13)
CHECK FOR CONVERGENCE
IF( ALCONV ) GO TO 20

IF( LEV .EQ. L .AND. NCYC .EQ. MAXCYC ) GO TO 40

INITIATE COARSE GRID CORRECTION

IF( LIST .NE. O ) WRITE(UNIT,100)
FORMAT(1H ' STARTING COARSE GRID CORRECTION',13)
CALL CGCST(MEMORY,LEV,ALPHA,DELTA,EXTRAP,LST,UNIT)
IF( LEV .EQ. L ) NCYC = NCYC + 1
LEV = LEV - 1
MINSWP = PSWP
GO TO 10

HAS CONVERGENCE AT LEVEL L BEEN ACHIEVED ?
SUBROUTINE SOLVE

Calls relaxation routine to apply minimum of MINSWP relaxation sweeps on all grids at level L, or invokes direct solver if L is coarsest level. Returns when problem has been solved on level L, or if convergence is slow indicating the need for a coarse grid correction.

Parameters:

MEMORY  ->  Main storage array
L  ->  Level that problem is being solved at
ALCONV  ->  Returns .TRUE. if problem is solved on all grids at level L. .FALSE. if not
ETA  ->  Eta
INIMOD  ->  Initial relaxation mode
MINSWP  ->  Minimum number of relaxation sweeps
NSWP  ->  Returns number of sweeps performed
LIST  ->  Listing flag
UNIT  ->  Listing unit

Routines called:

MAXO
LINRLX
DIRECT
SUBROUTINE SOLVE(MEMORY,L,ALCONV,ETA,INIMOD,MINSWP,NSWP,
                 LIST,UNIT)

C DATA STRUCTURE

C INTEGER GFSTR(50,10), XFSTR(50,10), YFSTR(50,10)
C INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
  SON(50), NPT(50), NX(50), NY(50), XHIB(50),
  XLOB(50), YHIB(50), YLOB(50)
C REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
  XINC(50), YINC(50)
C LOGICAL CONV(50), SLOW(50)
C INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
C INTEGER MEMPT
C INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU
C INTEGER NULL / ZBOBOBOBOBO /
C COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHR, SON,
  NX, NY, NPT, AREA, XINC, YINC,
  XFSTR, YFSTR, GFSTR, XLOB, XHIB,
  YLOB, YHIB, ERROR, CONV, SLOW, EPSI,
  NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT
C COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS,
  TAU, KSQR, NXFUNC, NYFUNC, NGFUNC
C REAL*8 MEMORY(1)
C REAL*8 DIRECT, LINRLX
C REAL*8 ERRGRD, DIREPS, ETA, INFNTY
C INTEGER NXTMOD(4), SWPCNT(50)
C INTEGER MAXO
C INTEGER GRID, INIMOD, L, LIST, MINSWP, MODE, NSWP, NTEMP,
  NXGRID, NYGRID, UNIT
C INTEGER DELPHI, JACOB, MNITER, MXITER, NJ, NXW, NYW, RC,
  WORK
C LOGICAL ALSLOW, ALCONV
C COMMON / DIR / JACOB, NJ, WORK, NXW, NYW, DELPHI
  / COMSWP / SWPCNT
C DATA INFNTY / 1.D50 /,
  MNITER / 3 /,
  MXITER / 10 /,
  NXTMOD / 2, 1, 3, 3/
WRITE(UNIT,2000) (EPSI(I),I=1,L)

2000 FORMAT(8E14.5)

GRID = LHEAD(L)
NSWP = 0
MODE = INIMOD

10 IF( GRID .EQ. NULL ) GO TO 20
    SLOW(GRID) = .FALSE.
    CONV(GRID) = .FALSE.
    ERROR(GRID) = INFNTY
    GRID = LLINK(GRID)
    GO TO 10

20 ALSLOW = .TRUE.
    ALCONV = .TRUE.
    GRID = LHEAD(L)

30 IF( GRID .EQ. NULL ) GO TO 50
    IF( NSWP .LT. MINSWP ) GO TO 32
    IF( SLOW(GRID) .AND. L .NE. CRSLEV ) GO TO 40
    IF( CONV(GRID)) GO TO 40

32 NXGRID = NX(GRID)
    NYGRID = NY(GRID)

C IF( L .EQ. CRSLEV ) GO TO 33
C
C PERFORM RELAXATION SWEEP
C
ERRGRD = LINRLX(MEMORY(GFSTRT(GRID,U)),
* MEMORY(GFSTRT(GRID,KSQR)),
* MEMORY(GFSTRT(GRID,RHS)),
* MEMORY(XFSTRT(GRID,R)).
* MEMORY(XFSTRT(GRID,RP)).
* MEMORY(XFSTRT(GRID,RM)).
* MEMORY(YFSTRT(GRID,W)).
* MEMORY(YFSTRT(GRID,WP)).
* MEMORY(YFSTRT(GRID,WM)).
* XINC(GRID),YINC(GRID),NXGRID,NYGRID,
* XLOB(GRID),XHIB(GRID),YLOB(GRID),YHIB(GRID).
* MEMORY(MEMPT),MEMORY(MEMPT+NTEMP),
* MEMORY(MEMPT+2*NTEMP),MEMORY(MEMPT+3*NTEMP),
* MODE,LIST,UNIT)

C SWPCT(L) = SWPCT(L) + 1
C
GO TO 34
C
C SOLVE DIRECTLY
C
DIREPS = EPSI(GRID)

33 ERRGRD = DIRECT(MEMORY(GFSTRT(GRID,U)),
* MEMORY(GFSTRT(GRID,KSQR)),
* MEMORY(GFSTRT(GRID,RHS)).
* MEMORY(MEMPT),
* MEMORY(XFSTRT(GRID,R)).
* MEMORY(XFSTRT(GRID,RP)),
* MEMORY(XFSTRT(GRID,RM)),
* MEMORY(YFSTRT(GRID,W)),
* MEMORY(YFSTRT(GRID,WP)),
* MEMORY(YFSTRT(GRID,WM))),
* MEMORY(JACOB),MEMORY(WORK),MEMORY(DELPHI),
* XINC(GRID),YINC(GRID),NXGRID,NYGRID,NJ,NXW,NYW,
* XLOB(GRID),XHIB(GRID),YLOB(GRID),YHIB(GRID),
* DIREPS,MINITER,MAXITER,LIST,UNIT,RC)

C SWPCNT(L) = SWPCNT(L) + 1
1000 IF( RC .NE. 0 ) WRITE(UNIT,1000)
     FORMAT(‘-ABORT PROGRAM: FAILURE IN DIRECT’) 
     EXIT
     RETURN

C 34 IF( ERRGRD .LE. EPSI(GRID) ) CONV(GRID) = .TRUE.
     SLOW(GRID) = .FALSE.
     IF( (ERRGRD / ERROR(GRID)) .GT. ETA )
     SLOW(GRID) = .TRUE.
     ERROR(GRID) = ERRGRD

C 40 ALSLOW = ALSLOW .AND. SLOW(GRID)
C ALCONV = ALCONV .AND. CONV(GRID)
C GRID = LLINK(GRID)
C GO TO 30
C 50 IF( ALCONV ) RETURN
C NSWP = NSWP + 1
C MODE = NXTMOD(MODE)
C IF( L .EQ. CRSLEV .OR. .NOT.ALSLOW .OR. NSWP .LT. MINSWP ) GO TO 20
C RETURN
C
C END

------------------------------------------------------------------------
C SUBROUTINE CGCST
C
C Starts coarse grid correction of level LEV on level LEV-1.
C Updates right-hand-side of level LEV-1 equations, calculates
C relative truncation error estimates, and updates convergence
C criteria for level LEV-1 and possibly level LEV.

C Parameters:
C
C MEMORY -> Main storage array
LEV ----> Level being corrected
ALPHA --> Alpha
DELTA --> Delta
EXTRAP --> Tau-extrapolation parameter
LIST ----> Listing flag
UNIT ----> Listing unit

Routines called:
COPY
INIT
OPERAT
INJECT
CLCTAU
NEWRHS

-----------------------------

SUBROUTINE CGCST(MEMORY,LEV,ALPHA,DELTA,EXTRAP,LIST,UNIT)

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)

INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
*     SON(50), NPT(50), NX(50), NY(50), XHIB(50),
*     XLOB(50), YHIB(50), YLOB(50)

REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
*     XINC(50), YINC(50)

LOGICAL CONV(50), SLOW(50)

INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC

INTEGER MEMPT

INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU

INTEGER NULL / Z80808080 /

COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHR, SON,
*     NX, NY, NPT, AREA, XINC, YINC,
*     XFSTRT, YFSTRT, GFSTRT, XLOB, XHIB,
*     YLOB, YHIB, ERROR, CONV, SLOW, EPSI,
*     NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT

COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS,
*     TAU, KSQR, NXFUNC, NYFUNC, NGFUNC

REAL*8 MEMORY(1)

REAL*8 CLCTAU

REAL*8 ALPHA, BETA, EXTRAP, MAXERR, NOTDEF, TAREA

INTEGER DAD, DADLEV, FSTBRN, GRID, LEV, LIST, NXD,
NYD, TAULEV, TEMP, TEMP2, UNIT

DATA NOTDEF / Z8080808080808080 /

OMEMPT = MEMPT
DADLEV = LEV - 1
DAD = LHEAD(DADLEV)

FOR EACH GRID ON LEVEL LEV - 1

IF( DAD .EQ. NULL ) GO TO 50
MEMPT = OMEMPT
FSTBRN = SON(DAD)

IF GRID HAS ANY SONS

IF( FSTBRN .EQ. NULL ) GO TO 40

THEN CREATE COPY OF CURRENT GRID FUNCTION AND ALLOCATE STORAGE

NXD = NX(DAD)
NYD = NY(DAD)
TAREA = 0.0DO
MAXERR = 0.0DO
GRID = FSTBRN
CALL COPY(MEMORY(GFSTRT(DAD,U)),
          MEMORY(GFSTRT(DAD,OLDU)),NXD,NYD)

TEMP1 = MEMPT
MEMPT = MEMPT + NPT(DAD)
CALL INIT(MEMORY(TEMP1),NXD,NYD,NOTDEF)

TEMP2 = MEMPT
MEMPT = MEMPT + NPT(DAD)
CALL INIT(MEMORY(TEMP2),NXD,NYD,0.0DO)

FOR EACH SON, APPLY DIFFERENCE OPERATOR, INJECT INTO TEMP1, AND INJECT (GRID,U) INTO (DAD,OLDU)

IF( GRID .EQ. NULL ) GO TO 30
CALL OPERAT(MEMORY(GFSTRT(GRID,U)),
            MEMORY(GFSTRT(GRID,KSQR)),MEMORY(MEMPT),
            MEMORY(MEMPT),
            MEMORY(XFSTRT(GRID,R)),MEMORY(XFSTRT(GRID,RP)),
            MEMORY(XFSTRT(GRID,RM)),MEMORY(YFSTRT(GRID,W)),
            MEMORY(YFSTRT(GRID,WP)),MEMORY(YFSTRT(GRID,WM)),
            XINC(GRID),YINC(GRID),NX(GRID),NY(GRID),
            XLOB(GRID),XHIB(GRID),YLOB(GRID),YHIB(GRID),
            O.DO,LIST,UNIT)

CALL INJECT(MEMORY(TEMP1),MEMORY(MEMPT),NXD,
            NYD,NX(GRID),NY(GRID),MEMORY(XFSTRT(DAD,R)),
            MEMORY(XFSTRT(DAD,W)),MEMORY(XFSTRT(GRID,R)),
            MEMORY(YFSTRT(GRID,W)),XINC(DAD),YINC(DAD),
            XINC(GRID),YINC(GRID),XLOB(DAD),XHIB(DAD),
            YLOB(DGRID),YHIB(GRID),LGRID)

CALL INJECT(MEMORY(GFSTRT(DAD,OLDU)),
            MEMORY(GFSTRT(GRID,U)),NXD,
            NYD,NX(GRID),NY(GRID),MEMORY(XFSTRT(DAD,R)),
            MEMORY(XFSTRT(GRID,U)),NXD,
            NYD,NX(GRID),NY(GRID),MEMORY(XFSTRT(DAD,R)))
* MEMORY(YFSTRT(DAD,W)), MEMORY(XFSTRT(GRID,R)),
* MEMORY(YFSTRT(GRID,W)), XINC(DAD), YINC(DAD),
* XINC(GRID), YINC(GRID), XLOB(DAD), YHIB(DAD),
* YLOB(DAD), YHIB(DAD), XLOB(GRID), XHIB(GRID),
* YLOB(GRID), YHIB(GRID), 0)
CALL INJECT(MEMORY(TEMP2),
* MEMORY(GFSTRT(GRID,RHS)), NXD,
* NYD, XN(GRID), YN(GRID), MEMORY(XFSTRT(DAD,R)),
* MEMORY(YFSTRT(DAD,W)), MEMORY(XFSTRT(GRID,R)),
* MEMORY(YFSTRT(GRID,W)), XINC(DAD), YINC(DAD),
* XINC(GRID), YINC(GRID), XLOB(DAD), XHIB(DAD),
* YLOB(DAD), YHIB(DAD), XLOB(GRID), XHIB(GRID),
* YLOB(GRID), YHIB(GRID), 1)
TAREA = TAREA + AREA(GRID)
IF( ERROR(GRID) .GT. MAXERR )
* MAXERR = ERROR(GRID)
GRID = BROTHR(GRID)
GO TO 20

C CALCULATE LOCAL TRUNCATION ERROR AND UPDATE
(DAD, RHS)

CALL OPERAT(MEMORY(GFSTRT(DAD,OLDU)),
* MEMORY(GFSTRT(DAD,KSQR)), MEMORY(MEMPT),
* MEMORY(MEMPT),
* MEMORY(XFSTRT(DAD,R)), MEMORY(XFSTRT(DAD,RP)),
* MEMORY(XFSTRT(DAD,RM)), MEMORY(YFSTRT(DAD,W)),
* MEMORY(XFSTRT(DAD,WP)), MEMORY(YFSTRT(DAD,W)),
* XINC(DAD), YINC(DAD), NXD, NYD,
* XLOB(DAD), XHIB(DAD), YLOB(DAD), YHIB(DAD),
* O.OOO, LIST, UNIT)
NRMTAU(DAD) = CLCTAU(MEMORY(GFSTRT(DAD,TAU)),
* MEMORY(MEMPT), MEMORY(TEMP1), MEMORY(XFSTRT(DAD,R)),
* MEMORY(YFSTRT(DAD,W)), NXD, NYD)
CALL COPY(MEMORY(GFSTRT(DAD,OLDU)),
* MEMORY(GFSTRT(DAD,U)), NXD, NYD)
CALL NEWRHS(MEMORY(GFSTRT(DAD,RHS)),
* MEMORY(GFSTRT(DAD,TAU)), MEMORY(TEMP2), NXD, NYD,
* DADLEV, FINLEV, EXTRAP)

UPDATE CONVERGENCE CRITERIA

EPSI(DAD) = DELTA * MAXERR *
* (TAREA / AREA(DAD))
WRITE(6,1000) DADLEV, EPSI(DAD)
1000 FORMAT('OCONVERGENCE CRITERIA ON LEVEL ',13,
* ' IS NOW ',E15.5)
IF( DADLEV .NE. (FINLEV - 1) ) GO TO 40
TGRID = SON(DAD)
35 IF( TGRID .EQ. NULL ) GO TO 40
EPSI(TGRID) = ALPHA * NRMTAU(DAD) *
* (AREA(DAD) / AREA(TGRID))
TGRID = BROTHR(TGRID)
GO TO 35

DAD = LLINK(DAD)
GO TO 10

MEMPT = OMEMPT
SUBROUTINE CGCFIN

 Finishes coarse grid correction on level LEV. Correction is interpolated to level LEV+1 and fine grid functions are updated appropriately.

 Parameters:
 MEMORY -> Main storage array
 LEV ---> Level that correction has been calculated on
 LIST ---> Listing flag
 UNIT ---> Listing unit

 Routines called:
 ADDSUB
 INIT
 INTERP

 SUBROUTINE CGCFIN(MEMORY,LEV,LIST,UNIT)

 GRID DATA STRUCTURE

 INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)
 INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
 * SON(50), NPT(50), NX(50), NY(50), XHIB(50),
 * XLOB(50), YHIB(50), YLOB(50)
 REAL*8 AREA(50), EPSI(50), ERRDR(50), NRMTAU(50),
 * XINC(50), YINC(50)
 LOGICAL CONV(50), SLOW(50)
 INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
 INTEGER MEMPT
 INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU
 INTEGER NULL / Z80808080 /

 COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHR, SON,
 * NX, NY, NPT, AREA, XINC, YINC,
 * XFSTRT, YFSTRT, GFSTRT, XLOB, XHIB,
 * YLOB, YHIB, ERRDR, CONV, SLOW, EPSI,
*                       NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT
C
* COMMON              / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS,
*                      TAU, KSQR, NXFUNC, NYFUNC, NGFUNC
C
REAL*8 MEMORY(1)
INTEGER DAD, FSTBRN, GRID, LEV, LIST, NXG, NYG, UNIT

C
C FOR EACH GRID ON LEVEL LEV
C
10 IF( DAD .EQ. NULL ) GO TO 40
   IF GRID HAS SONS
   FSTBRN = SON(DAD)
   IF( FSTBRN .EQ. NULL ) GO TO 30
   CALL ADDSUB(MEMORY(GFSTRT(DAD,U) ),
                MEMORY(GFSTRT(DAD,OLDU) ),
                MEMORY(GFSTRT(DAD,OLDU) ),
                NX(DAD),NY(DAD),-1.ODO)
   FOR EACH GRID WHICH IS A SON OF DAD, UPDATE GRID
   FUNCTION (GRID,U)
   GRID = FSTBRN
   20 IF( GRID .EQ. NULL ) GO TO 30
      NXG = NX(GRID)
      NYG = NY(GRID)
      CALL INIT(MEMORY(MEMPT ),NXG,NYG,0.ODO)
      CALL INTERP(MEMORY,DAD,GFSTRT(DAD,OLDU) ,
                  MEMPT,2,LIST,UNIT)
      CALL ADDSUB(MEMORY(GFSTRT(GRID,U) ),
                  MEMORY(MEMPT),MEMORY(GFSTRT(GRID,U) ),
                  NXG,NYG,1.ODO)
      GRID = BROTHR(GRID)
      GO TO 20
   30 DAD = LLINK(DAD)
   GO TO 10
C
40 RETURN
C
END
C
SUBROUTINE ADAPT
Scans truncation error estimate contained in TAUL2 to decide where outer boundary of new fine grid should be. (If new grid is at level L, truncation error estimate is for level L-2 relative to L-1)

Parameters:

- `TAUL2(NXL2,NYL2)` -> Truncation error estimate
- `XL2(NXL2)` -> X coordinates of level L-2
- `ALPHA` -> Alpha
- `CVTAU` -> Truncation error convergence criterion
- `AREAL1` -> No longer used - dummy parameter
- `AREAL` -> Coordinate of new outer boundary
- `NWXMXD` -> Returns .TRUE. if truncation error uniformly below CVTAU

```
SUBROUTINE ADAPT(NXL2,NYL2,TAUL2,XL2,ALPHA,CVTAU,AREAL1,AREAL,NWXMXD,NOREF)

INTEGER NXL2, NYL2
REAL*8 TAUL2(NXL2,NYL2)
REAL*8 XL2(NXL2)
REAL*8 DABS
REAL*8 ALPHA, AREAL, AREAL1, CVSM, CVTAU, ETAULJ, NOCVSM, NOTDEF, NWXMXD, SUMTAU, TAUIJ, TOTSM

INTEGER I, IMXNW, J, NXL2M1, NYL2M1, NYL2M2
LOGICAL NOREF

DATA NOTDEF / Z8080808080808080 /
DATA CVSM = 0.0D0
DATA NOCVSM = 0.0D0
NXL2M1 = NXL2 - 1
NYL2M1 = NYL2 - 1
NYL2M2 = NYL2 - 2
NOREF = .TRUE.
IMXNW = NXL2

I = NXL2M1

10 IF( I .LT. 2 ) GO TO 60
    SUMTAU = 0.0D0

    DO 20 J = 2, NYL2M1
             TAUIJ = TAUL2(I,J)
       IF( TAUIJ .EQ. NOTDEF ) GO TO 50
             SUMTAU = SUMTAU + DABS(TAUIJ)
```

10

If the truncation error is below the convergence criterion, the outer boundary is moved to the new grid. Otherwise, the process continues.
CONTINUE
C
ETAULJ = ALPHA * SUMTAU / NYL2M2
IF( ETAULJ .LE. CVTAU .AND. NOREF ) GO TO 40
C
REFINEMENT NECESSARY IN THIS REGION
C
IF( .NOT. NOREF ) GO TO 30
NOREF = .FALSE.
IF( I .NE. NXL2M1 ) IMXNW = I
NOCVSM = NOCVSM + ETAULJ
GO TO 50
C
NO REFINEMENT NEEDED YET
C
CVSM = CVSM + ETAULJ
C
50  
I = I - 1
C
GO TO 10
C
60  
TOTSM = CVSM + NOCVSM
NWAREA = XL2(IMXNW)
NWXMXD = XL2(IMXNW)
IF( IMXNW .EQ. 2 ) NOREF = .TRUE.
C
RETURN
C
END

FUNCTION CLCTAU

Calculates relative truncation error estimate from functions LIU and ILU and places in array TAU. Returns norm of TAU. Value of NOTDEF is placed in TAU anywhere where truncation error is undefined.

Parameters:

TAU(NX, NY) --> Relative truncation error estimate
LIU(NX, NY) --> Residuals of injected fine grid fcn
ILU(NX, NY) --> Injected fine grid residuals
X, Y --------> Grid coordinates (currently unused)

Routines called:

DABS

DOUBLE PRECISION FUNCTION CLCTAU(TAU, LIU, ILU, X, Y, NX, NY)
INTEGER NX, NY
REAL*8 ILU(NX,NY), LIU(NX,NY), TAU(NX,NY)
REAL*8 X(1), Y(1)
REAL*8 DABS
REAL*8 ILUXY, NOTDEF, NRMTAU, TAUXY
INTEGER INRM, IX, IY, NX1, NY1
DATA NOTDEF / Z8080808080808080 /
NRMTAU = 0.0D0
DO 30 IY = 1 , NY
   DD DO 20 IX = 1 , NX
      ILUXY = ILU(IX,IY)
      IF ( ILUXY .EQ. NOTDEF ) GO TO 10
      TAUXY = LIU(IX,IY) - ILUXY
      TAU(IX,IY) = TAUXY
      GO TO 20
   10 CONTINUE
   20 TAU(IX,IY) = NOTDEF
30 CONTINUE
   NX1 = NX - 1
   NY1 = NY - 1
DO 50 IX = 2 , NX1
   DO 40 IY = 2 , NY1
      NRMTAU = NRMTAU + DABS(TAU(IX,IY))
40 CONTINUE
50 CONTINUE
CLCTAU = NRMTAU / ((NX1 - 1) * (NY1 - 1))
RETURN
END

SUBROUTINE NEWRHS
Calculates new right-hand-side of difference equations from truncation error estimate, TAU and injected right-hand-side, INJRHS of finer level difference equations.
Parameters:
SUBROUTINE NEWRHS(RHS, TAU, INJRHS, NX, NY, L, LASTL, EXTRAP)

INTEGER NX, NY
REAL*8 INJRHS(NX, NY), RHS(NX, NY), TAU(NX, NY)
REAL*8 EXTRAP, NOTDEF, TAUEXT, TAUXY
INTEGER IX, IY, L, LASTL

DATA NOTDEF / 0.0D0 /

IF (L .EQ. (LASTL - 1)) GO TO 10
   TAUEXT = 1.0D0
   GO TO 20
10   TAUEXT = EXTRAP
20   DO 40 IY = 1, NY
   DO 30 IX = 1, NX
      TAUXY = TAU(IX, IY)
      IF (TAUXY .EQ. NOTDEF) GO TO 25
      RHS(IX, IY) = TAUEXT * TAUXY + INJRHS(IX, IY)
      GO TO 30
25   TAU(IX, IY) = 0.0D0
30   CONTINUE
40   CONTINUE

RETURN

END
FUNCTION LINRLX

Line relaxation routine. Performs one line relaxation
sweep of unknown function PHI. If MODE .GT. 2, then lines
of constant radial coordinate are used; If MODE .LE. 2,
lines of constant angular coordinate are used. If MODE is
odd, forward sweeping direction is used; If MODE is even
reverse sweeping direction is used. Uses external routine
TRISLV (MTS library function) to solve tridiagonal systems.
Invokes appropriate routines to smooth boundary difference
equations. Returns norm of interior dynamic residuals.

Parameters:

PHI(NS,NW) --> Unknown function
KSQR(NS,NW) --> Extrinsic curvature squared
RHS(NS,NW) --> Right-hand-side of difference system
S, S2, S4 --> Radial coordinate functions
W, WP, WM --> Angular
DELS --> Radial mesh spacing
DELW --> Angular
XLOB, XHIB, --> Boundary flags
YLOB, YHIB --> Angular
LOWER --> Lower diagonal of tridiagonal system
DIAG --> Main
UPPER --> Upper
RESRHS --> Right-hand-side of PHI
Also used for update of PHI
MODE --> Selects line grouping/sweeping direction
LIST --> Listing flag
UNIT --> Listing unit

Routines called:

NRMACC
NRMCLC
TRISLV
RXLOB
RYB

DOUBLE PRECISION FUNCTION LINRLX(PHI,KSQR,RHS,
* S,S2,S4,W,WP,WM,DELS,
* DELW,NS,NW,XLOB,XHIB,YLOB,YHIB,LOWER,
* DIAG,UPPER,RESRHS,MODE,LIST,UNIT)
INTEGER NS, NW
REAL*8 KSOR(NS, NW), PHI(NS, NW), RHS(NS, NW)
REAL*8 DIAG(1), LOWER(1), RESRHS(1), S(1), S2(1),
     S4(1), UPPER(1), W(1), WM(1), WP(1)
REAL*8 NRMACC, NRMCLC
REAL*8 C1, C1A, C2, C3, D1, D2, DELS, DELSM2, DELW,
     DELWM2, KPHIM7, KSORIJ, PHIJJ, PHIM7, RES, S2I,
     S4I, WMJ, WPJ
INTEGER I, IFIN, IST, ISTEP, ITRI, J, JFIN, JST, JSTEP,
     LIST, MODE, NPT, NSM1, NTRI, NWM1, UNIT, XBST,
     XBFIN, XHIB, XLOB, XBST, YBFIN, YHIB, YLOB

LINRLX = 0.000
NSM1 = NS - 1
NWM1 = NW - 1
NPT = (NS - 2) * (NW - 2)
DELSM2 = 1.000 / (DELS ** 2)
DELWM2 = 1.000 / (DELW ** 2)

INTERIOR RELAXATION
IF( MODE .GT. 2 ) GO TO 60
LINES OF CONSTANT ANGULAR COORDINATE
NTRI = NS - 2
IF( MODE .EQ. 2 ) GO TO 10
FORWARD SWEEPING DIRECTION
JST = 2
JFIN = NWM1
JSTEP = 1
GO TO 20
REVERSE SWEEPING DIRECTION
10
JST = NWM1
JFIN = 2
JSTEP = -1
20
J = JST
START LOOP
30
WPJ = WP(J)
WMJ = WM(J)
ITRI = 1
DD 40 I = 2 , NSM1
KSORIJ = KSOR(I, J)
PHIIJ = PHI(I, J)
PHIM7 = 1.000 / PHIJI ** 7
KPHIM7 = KSRRIJ * PHIM7
C1 = S4(I) * DELSM2
S2I = S2(I)
C2 = S2I * DELWM2 * WPJ
C3 = S2I * DELWM2 * WMJ
D1 = C1 * (PHI(I+1,J) - 2.000 * PHIJI + PHI(I-1,J))
D2 = C2 * (PHI(I,J+1) - PHIJI) + C3 * (PHI(I,J-1) - PHIJI)
RES = D1 + D2 + 0.12500 * KPHIM7 - RHS(I,J)
LINRLX = NRMACC(LINRLX,RES,0.000,0.000)

C SET UP TRIDIAGONAL SYSTEM
C
        DIAG(ITRI) = -(2.000 * C1 + C2 + C3 + 0.87500 * KPHIM7 / PHIJI)
        UPPER(ITRI) = C1
        LOWER(ITRI) = C1
        RESRHS(ITRI) = RES
        ITRI = ITRI + 1

C CONTINUE
C
CALL TRISLV(NTRI, LOWER, DIAG, UPPER, RESRHS, 0.8300)
ITRI = 1
C
C UPDATE PHI
C
DO 50 I = 2, NSM1
   PHI(I,J) = PHI(I,J) - RESRHS(ITRI)
   ITRI = ITRI + 1
50 CONTINUE
C
C END LOOP
C
IF( J .EQ. JFIN ) GO TO 120
J = J + JSTEP
GO TO 30
C
C LINES OF CONSTANT RADIAL COORDINATE
C
60 NTRI = NW - 2
IF( MODE .EQ. 4 ) GO TO 70
C
C FORWARD SWEEPING DIRECTION
C
IST = 2
IFIN = NSM1
ISTEP = 1
GO TO 80
C
C REVERSE SWEEPING DIRECTION
C
70 IST = NSM1
IFIN = 2
ISTEP = -1
START LOOP

S2I = S2(I)
S4I = S4(I)
C1 = S4I * DELSM2
C1A = S2I * DELWM2
ITRI = 1

DO 100 J = 2, NWM1
KSORIJ = KSQR(I,J)
PHHIJ = PHI(I,J)
PHIM7 = 1.0D0 / PHHIJ ** 7
KPHIM7 = KSORIJ * PHIM7
C2 = C1A * WP(J)
C3 = C1A * WM(J)
D1 = C1A * (PHI(I+1,J) - 2.0DO * PHHIJ + PHI(I-1,J))
D2 = C2 * (PHI(I,J+1) - PHHIJ) + C3 * (PHI(I,J-1) - PHHIJ)
RES = D1 + D2 + 0.125DO * KPHIM7 - RHS(I,J)
LINRLX = NRMACC(LINRLX,RES,0.0DO,0.0DO)

SET UP TRIDIAGONAL SYSTEM

DIAG(ITRI) = -(2.0DO * C1 + C2 + C3 + 0.875DO * KPHIM7 / PHHIJ)
UPPER(ITRI) = C2
LOWER(ITRI) = C3
RESRHS(ITRI) = RES
ITRI = ITRI + 1

CONTINUE

SOLVE TRIDIAGONAL SYSTEM

CALL TRISLV(NTRI,LOWER,DIAG,UPPER,RESRHS,0,8.300)
ITRI = 1

UPDATE PHI

DO 110 J = 2, NWM1
PHI(I,J) = PHI(I,J) - RESRHS(ITRI)
ITRI = ITRI + 1

CONTINUE

END LOOP

IF( I .EQ. IFIN ) GO TO 120
I = I + ISTEP
GO TO 90

LINRLX = NRMCLC(LINRLX,NPT)
IF( LIST .NE. 0 ) WRITE(UNIT,200) LINRLX
200 FORMAT(’ INTERIOR RESIDUAL: ’,T60,E12.4)

BOUNDARY RELAXATIONS

XBST = 1
XBFIN = NS - 1
YBST = 1
YBFIN = NW
IF( XLOB .NE. 0 ) XBST = 2
IF( YLOB .NE. 0 ) YBST = 2
IF( YHIB .NE. 0 ) YBFIN = NW - 1

IF( XLOB .NE. 0 ) RES = RXLOB(PHI,RHS,NS,NW,XLOB,1,DELS,
    YBST,YBFIN,O.9DO,3,LIST,UNIT)
* IF( YLOB .NE. 0 ) RES = RYB(PHI,RHS,NS,NW,YLOB,1,DELW,
    XBST,XBFIN,O.5DO,3,LIST,UNIT)
* IF( YHIB .NE. 0 ) RES = RYB(PHI,RHS,NS,NW,YHIB,NW,DELW,
    XBST,XBFIN,O.5DO,3,LIST,UNIT)

RETURN

C
C ABNORMAL EXITS
C
300 WRITE(UNIT,310)
310 FORMAT('OS/R LINRLX: ERROR IN EXT. ROUTINE TRISLV')
RETURN

C
C END

C
C FUNCTION RXLOB
C
C Performs NSWP smoothing sweeps of inner x boundary
difference equations. Returns norm of residuals.
Sweeping direction controlled by value of MODE.

Parameters: See FUNCTION LINRLX

C
C DOUBLE PRECISION FUNCTION RXLOB(PHI,RHS,NX,NY,NSWP,BX,DELX,
    YBST,YBFIN,ETA,MODE,LIST,UNIT)
C
C INTEGER NX, NY
C REAL*8 PHI(NX,NY), RHS(NX,NY)
C REAL*8 DABS
C REAL*8 DELXM2, DELX, ETA, RFINNW, RPOS, RPOSM, RPOSNW,
    RPOSP, RST, RSTNW, RSTP, RTOT
C INTEGER START(3)
C INTEGER BX, BX1, BX2, IFIN, IST, LIST, MODE, N, NSWP, POS,
    ST, STEP, SWEEP, UNIT, YBFIN, YBST
N = YBFIN - YBST + 1
DELXM2 = -0.5D0 / DELX
BX1 = BX + 1
BX2 = BX + 2

IST = YBST + 1
IFIN = YBFIN - 1

GO TO( 10, 20, 30 ), MODE

10
ST = YBST
STEP = 1
GO TO 40

20
ST = YBFIN
STEP = -1
GO TO 40

30
ST = YBST
STEP = 1
START(1) = YBST
START(3) = YBFIN

40 SWEEP = 0

50 RTOT = 0.0D0
RST = DELXM2 * (PHI(BX,ST) - PHI(BX2,ST)) +
* 0.5D0 * PHI(BX1,ST) - RHS(BX,ST)
RSTP = DELXM2 * (PHI(BX,ST+STEP) - PHI(BX2,ST+STEP)) +
* 0.5D0 * PHI(BX1,ST+STEP) - RHS(BX,ST+STEP)
RSTNW = ETA * RST
RTOT = RTOT + DABS(RSTNW)
PHI(BX,ST) = PHI(BX,ST) + (RSTNW - RST) / DELXM2
POS = ST + STEP
RPOS = RSTP
RPOSM = RSTNW

60 DO 1 I = IST , IFIN
    RPOSP = DELXM2 * (PHI(BX,POS+STEP) - PHI(BX2,POS+STEP)) +
    * 0.5D0 * PHI(BX1,POS+STEP) - RHS(BX,POS+STEP)
    RPOSNW = 0.5D0 * (RPOSM + RPOSP)
    RTOT = RTOT + DABS(RPOSNW)
    PHI(BX,POS) = PHI(BX,POS) + (RPOSNW - RPOS) / DELXM2
    RPOSM = RPOSNW
    RPOS = RPOSP
    POS = POS + STEP
    CONTINUE

65 RFINNW = ETA * RPOS
RTOT = RTOT + DABS(RFINNW)
PHI(BX,POS) = PHI(BX,POS) + (RFINNW - RPOS) / DELXM2

RXLOB = RTOT / N
IF( LIST .EQ. 0 ) GO TO 65
    WRITE(UNIT,100) RXLOB
100 FORMAT( ' INNER BOUNDARY RESIDUAL: ',T60,E12.4)

65 SWEEP = SWEEP + 1
IF( SWEEP .GE. NSWP ) GO TO 70
C
IF( MODE .NE. 3 ) GO TO 50
ST = START(STEP+2)
STEP = -STEP
GO TO 50
C
70 RETURN
C
C
END
C
C
FUNCTION RYB
C
C
Performs NSWP smoothing sweeps of y boundary difference equations. Returns norm of residuals. Boundary swept controlled by value of BY. Sweeping direction controlled by value of MODE.
Parameters: See FUNCTION LINRLX
C
C
DOUBLE PRECISION FUNCTION RYB(PHI,RHS,NX,NY,NSWP,BY,DELY,
*     XBST,XBFIN,ETA,MODE,LIST,UNIT)
C
INTEGER NX, NY
C
REAL*8 PHI(NX,NY), RHS(NX,NY)
C
REAL*8 DABS
C
REAL*8 DELYM2, DELY, ETA, RFINNW, RPOS, RPOSM, RPOSNW,
*     RPOSP, RST, RSTNW, RSTP, RTOT
C
INTEGER START(3)
C
INTEGER BY, BY2, IFIN, IST, LIST, MODE, N, NSWP, POS, ST,
*     STEP, SWEEP, UNIT, XBFIN, XBST
C
N = XBFIN - XBST + 1
DELYM2 = 0.5D0 / DELY
IF( BY .EQ. NY ) GO TO 3
BY2 = 3
DELYM2 = -DELYM2
GO TO 5
BY2 = NY - 2
C
3
IST = XBST + 1
IFIN = XBFIN - 1
GO TO( 10, 20, 30 ), MODE
C
10  ST = XBST
    STEP = 1
    GO TO 40
C
20  ST = XBFIN
    STEP = -1
    GO TO 40
C
30  ST = XBST
    STEP = 1
    START(1) = XBST
    START(3) = XBFIN
C
40  SWEEP = 0
C
50  RTOT = 0.0D0
    RST = DELYM2 * (PHI(ST,BY) - PHI(ST,BY2)) - RHS(ST,BY)
    RSTP = DELYM2 * (PHI(ST+STEP,BY) - PHI(ST+STEP,BY2)) - 
    RHS(ST+STEP,BY)
    RSTNW = ETA * RST
    RTOT = RTOT + DABS(RSTNW)
    PHI(ST,BY) = PHI(ST,BY) + (RSTNW - RST) / DELYM2
    POS = ST + STEP
    RPOS = RSTP
    RPOSM = RSTNW
C
60  DO 60 I = 1ST, IFIN
    RPOSP = DELYM2 * (PHI(POS+STEP,BY) - PHI(POS+STEP,BY2)) - 
    RHS(POS+STEP,BY)
    RPOSNW = 0.5D0 * (RPOSM + RPOSP)
    RTOT = RTOT + DABS(RPOSNW)
    PHI(POS,BY) = PHI(POS,BY) + (RPOSNW - RPOS) / DELYM2
    RPOSM = RPOSNW
    RPOS = RPOSP
    POS = POS + STEP
    CONTINUE
C
62  RFINNW = ETA * RPOS
    RTOT = RTOT + DABS(RFINNW)
    PHI(POS,BY) = PHI(POS,BY) + (RFINNW - RPOS) / DELYM2
C
65  RYB = RTOT / N
    IF( LIST .EQ. 0 ) GO TO 65
    IF( BY .EQ. NY ) GO TO 62
    WRITE(UNIT,100) RYB
700  GO TO 65
62  WRITE(UNIT,110) RYB
110  FORMAT( ' TOP BOUNDARY RESIDUAL:',T60,E12.4)
C
65  SWEEP = SWEEP + 1
C
69  IF( SWEEP .GE. NSWP ) GO TO 70
C
6A  IF( MODE .NE. 3 ) GO TO 50
    ST = START(STEP+2)
    STEP = -STEP
    GO TO 50
SUBROUTINE OPERAT

Applies interior difference equations to unknown function PHI, placing residuals in array RESID. (Right-hand-side of difference equations may be subtracted by setting RHSSW to 1.0DO - else should be 0.0DO) Calls appropriate routines to calculate boundary residuals, if necessary.

Parameters:

PHI(NS,NW) --> Unknown function
KSQR(NS,NW) --> Extrinsic curvature squared
RHS(NS,NW) --> Right-hand-side of difference system
RESID(NS,NW) --> Residual array
S, S2, S4 --> Radial coordinate functions
W, WP, WM --> Angular
DELS --> Radial mesh spacing
DELW --> Angular
XLOB, XHIB, YLOB, YHIB --> Boundary flags
RESRHS --> Enables/disables subtraction of RHS
LIST --> Listing flag
UNIT --> Listing unit

Routines called:

VALURC
OPXLOB
OPYB

SUBROUTINE OPERAT(PHI, KSQR, RHS, RESID, S, S2, S4, W, WP, WM, DELS, * DELW, NS, NW, XLOB, XHIB, YLOB, YHIB, RHSSW, LIST, UNIT)

INTEGER NS, NW

REAL*8 KSQR(NS,NW), PHI(NS,NW), RHS(NS,NW), RESID(NS,NW)
REAL*8 S(1), S2(1), S4(1), W(1), WM(1), WP(1)
REAL*8 C1, C2, C3, D1, D2, DELS, DELSM2, DELW, DELWM2, PHIJJ, RHSSW, S2I, WMJ, WPJ
INTEGER I, J, LIST, NSM1, NWM1, UNIT, XHIB, XLOB, YHIB, YLOB
IF ( LIST .GT. 0 ) WRITE(UNIT,200) NS, NW
200 FORMAT( ' ENTERING OPERAT - ',I4, ' X ',I4, ' GRID')
C
NSM1 = NS - 1
NWM1 = NW - 1
DELSM2 = 1.0DO / (DELS ** 2)
DELWM2 = 1.0DO / (DELW ** 2)
C
DO 20 J = 2 , NWM1
  WPJ = WP(J)
  WMJ = WM(J).
C
DO 10 I = 2 , NSM1
  PHIIJ = PHI(I,J)
  C1 = S4(I) * DELSM2
  S2I = S2(I)
  C2 = S2I * DELWM2 * WPJ
  C3 = S2I * DELWM2 * WMJ
  D1 = C1 * (PHI(I+1,J) - 2.0DO * PHIIJ + PHI(I-1,J))
  D2 = C2 * (PHI(I,J+1) - PHIIJ) +
  * C3 * (PHI(I,J-1) - PHIIJ)
  RESID(I,J) = D1 + D2 + 0.125DO * KSOR(I,J) / PHIIJ ** 7
  * - RHS(I,J) * RHSSW
10 CONTINUE
C
20 CONTINUE
C
IF( XLOB .NE. 0 ) GO TO 30
  CALL VALURC(RESID,NS,NW,1.0DO)
GO TO 40
30 CALL OPXLOB(PHI,RHS,RESID,DELS,NS,NW,RHSSW)
40 CALL VALURC(RESID,NS,NW,0.0DO)
IF( YLOB .NE. 0 ) GO TO 50
  CALL VALURC(RESID,NS,NW,1.0DO)
GO TO 60
50 CALL OPYB(PHI,RHS,RESID,DELW,NS,NW,1,RHSSW)
60 IF( YHIB .NE. 0 ) GO TO 70
  CALL VALURC(RESID,NS,NW,1.0DO)
GO TO 80
70 CALL OPYB(PHI,RHS,RESID,DELW,NS,NW,NW,RHSSW)
80 RETURN
C
END
C
SUBROUTINE OPXLOB
C
Applies inner x boundary difference equations to unknown
function PHI
C
Parameters: See SUBROUTINE OPERAT
C
-------
SUBROUTINE OXLOB(PHI, RHS, RESID, DELS, NS, NW, RHSSW)

INTEGER NS, NW

REAL*8 PHI(NS, NW), RHS(NS, NW), RESID(NS, NW)

REAL*8 DELS, DELS2M, RHSSW

INTEGER J

DELS2M = 0.5D0 / DELS

DO 10 J = 1, NW
   RESID(1, J) = DELS2M * (PHI(3, J) - PHI(1, J)) +
                  0.5DO * PHI(2, J) - RHS(1, J) * RHSSW
10 CONTINUE

RETURN

END

SUBROUTINE OYB

Applies y boundary difference equations to unknown function PHI

Parameters: see SUBROUTINE OPERAT

SUBROUTINE OYB(PHI, RHS, RESID, DELW, NR, NW, BY, RHSSW)

INTEGER NR, NW

REAL*8 PHI(NR, NW), RHS(NR, NW), RESID(NR, NW)

REAL*8 DELW, DELW2, RHSSW, SIGN

INTEGER BY, BY2, I

DELW2 = 0.5DO / DELW

IF( BY .EQ. 1 ) GO TO 10
   BY2 = NW - 2
   SIGN = 1.0DO
GO TO 20
10 BY2 = 3
   SIGN = -1.0DO
20 CONTINUE
C
DO 30 I = 1 , NR
  RESID(I,BY) = SIGN * (PHI(I,BY) - PHI(I,BY2)) * DELW2 -
    * RHS(I,BY) * RHSSW
30    CONTINUE
C
RETURN
C
C
END
SUBROUTINE DIRECT

Routine for solving difference equations using Newton-direct method. Performs minimum of MNITER Newton iterations, until residual norm is less than EPSI or MXITER iterations have been completed. Calls external CHLSKY to solve linear system.

Parameters:

PHI(NX,NY) --> Unknown function
KSQR(NX,NY) --> Extrinsic curvature squared
RHS(NX,NY) --> Right-hand-side of difference system
RESID(NX,NY) --> Residual array
S, S2, S4 --> Radial coordinate functions
W, WP, WM --> Angular
JACOB(NJ,NJ) --> Jacobian array
WORK(NXW,NYW) --> Work array for inversion of JACOB
DELPHI --> Function update array
DELW --> Angular
XLOB, XHIB, YLOB, YHIB --> Boundary flags
EPSI --> Convergence criterion
MNITER --> Minimum number of Newton iterations
MXITER --> Maximum
LIST --> Listing flag
UNIT --> Listing unit
RC --> .EQ. 0 --> Solution successful
       .NE. 0 --> No convergence after MXITER iterations

Routines called:

OPERAT
FDIAG
CHLSKY

DOUBLE PRECISION FUNCTION DIRECT(PHI,KSQR,RHS,RESID,
                                    S,S2,S4,W,WP,WM,
                                    JACOB,WORK,DELPHI,DELS,DELW,NX,NY,NJ,
                                    NXW,NYW,XLOB,XHIB,YLOB,YHIB,EPSI,MNITER,
                                    MXITER,LIST,UNIT,RC)

INTEGER NJ, NX, NXW, NY, NYW
REAL*8 DABS
REAL*8 JACOB(NJ,NJ), KSQR(NX,NY), PHI(NX,NY), RESID(NX,NY), RHS(NX,NY), WORK(NXW,NYW)
REAL*8 DELPHI(1), S(1), S2(1), S4(1), W(1), WM(1), WP(1)
REAL*8 DELS, DELW, EPSI, RES, RTOT
INTEGER FX, FY, I, ITER, IX, J, LIST, MXITER, NINT, NXM1, NXM2, NYM1, POS, RC, UNIT, XHIB, XLOB,
YHIB, YLOB
NXM1 = NX - 1
NXM2 = NX - 2
NYM1 = NY - 1
NINT = (NY - 2) * NXM2
ITER = 0

MAIN LOOP ...

CALCULATE RESIDUALS AND PLACE IN WORK ARRAY
10 CALL OPERAT(PHI,KSQR,RHS,RESID,S,S2,S4,W,WG,WG,DELS,
DELW,NX,NY,XLOB,XHIB,YLOB,YHIB,1.0D0,LIST,
UNIT)
FX = NXM1
POS = 1
RTOT = 0.0D0
DO 30 IX = 1, NXM2
20 DO 20 FY = 1, NY
RES = RESID(FX,FY)
WORK(POS,NYW) = RES
IF( FY .EQ. 1 .OR. FY .EQ. NY ) GO TO 25
RTOT = RTOT + DABS(RES)
POS = POS + 1
CONTINUE
FX = FX - 1
30 CONTINUE
DO 30 FY = 2, NYM1
WORK(POS,NYW) = RESID(1,FY)
pos = POS + 1
CONTINUE

CALCULATE RESIDUAL NORM AND EXIT IF CONVERGED
DIRECT = RTOT / NINT
IF( LIST .NE. 0 ) WRITE(UNIT,1000) ITER, DIRECT
1000 FORMAT(' ITERATION: ',I3,T60,E12.4)
IF( DIRECT .LE. EPSI .AND. ITER .GE. MNITER ) GO TO 100

FILL OFF-DIAGONAL ELEMENTS OF WORK ARRAY
DO 60 I = 1 , NJ
   DO 50 J = 1 , NJ
      WORK(I,J) = JACOB(I,J)
   CONTINUE
50 CONTINUE
CONTINUE
DEFINE DIAGONAL ELEMENTS
CALL FDIAG(WORK,NXW, NYW, PHI, KSOR, NX, NY, S2, S4, WP, WM,
          DELS, DELW)
SOLVE SYSTEM OF LINEAR EQUATIONS
CALL CHLSKY(WORK, NXW, NYW, DELPHI)
UPDATE PHI
FX = NXM1
POS = 1
DO 80 IX = 1 , NXM2
   DO 70 FY = 1 , NY
      PHI(FX,FY) = PHI(FX,FY) - DELPHI(POS)
      POS = POS + 1
   CONTINUE
70 CONTINUE
FX = FX - 1
CONTINUE
80 CONTINUE
DO 90 FY = 2 , NYM1
   PHI(1,FY) = PHI(1,FY) - DELPHI(POS)
   POS = POS + 1
90 CONTINUE
NEwTON ITERATION COMPLETE ... ENOUGH DONE ???
ITER = ITER + 1
IF( ITER .GE. MXITER ) GO TO 110
... END MAIN LOOP
GO TO 10
100 RC = 0
C RETURN
110 RC = 1
C RETURN
C END
210

SUBROUTINE FOFFD

C Defines off-diagonal elements of Jacobian array JACOB for
C solution of difference system by direct means.
C
Parameters:

JACOB(N,N) ---> Jacobian array
S2, S4 ------> Radial coordinate functions
WP, WM ------> Angular "
DELS ------> Radial mesh spacing
DELM ------> Angular "

SUBROUTINE FOFFD(JACOB,N,NX,NY,S2,S4,WP,WM,DELS,DELM)

INTEGER N
REAL*8 JACOB(N,N)
REAL*8 S2(1), S4(1), WM(1), WP(1)
REAL*8 C1FX, C2FX, DELS, DELS2M, DELSM1, DELSM2,
       DELW, DELW2M, DELWM1, DELWM2

INTEGER IX, IY, FX, FY, NX, NXM1, NXM2, NY, NYM1,
       NYM2, NYNYM1

NXM1 = NX - 1
NXM2 = NX - 2
NYM1 = NY - 1
NYM2 = NY - 2
NYNYM1 = NY + NYM1
DELSM1 = 1.0DO / DELS
DELSM2 = DELSM1 ** 2
DELS2M = 0.5DO * DELSM1
DELWM1 = 1.0DO / DELW
DELWM2 = DELWM1 ** 2
DELW2M = 0.5DO * DELWM1

ZERO JACOBIAN ARRAY

DO 10 I = 1, N
   DO 10 J = 1, N
      JACOB(I,J) = 0.0DO
   CONTINUE

FX = NXM1
POS = 1

DO 90 IX = 1, NXM2
   FY = 1
      C1FX = DELSM2 * S4(FX)

C
\[ C2FX = \text{DELWM2} \cdot S2(FX) \]

\[
\text{DO 80 IY = 1, NY}
\]
\[
\text{IF(IY .EQ. 1 ) GO TO 50}
\]
\[
\text{IF(IY .EQ. NY ) GO TO 60}
\]

**INTERIOR POINTS**

\[
\text{IF( IX .EQ. 1 ) GO TO 20}
\]
\[
\text{IF( IX .EQ. NXM2 ) GO TO 30}
\]

**CENTRAL INTERIOR POINTS**

\[
\text{JACOB(POS,POS+1) = C2FX \cdot WP(FY)}
\]
\[
\text{JACOB(POS,POS-1) = C2FX \cdot WM(FY)}
\]
\[
\text{JACOB(POS,POS+NY) = C1FX}
\]
\[
\text{JACOB(POS,POS-NY) = C1FX}
\]

**GO TO 40**

**RIGHT INTERIOR POINTS**

\[
\text{JACOB(POS,POS+1) = C2FX \cdot WP(FY)}
\]
\[
\text{JACOB(POS,POS-1) = C2FX \cdot WM(FY)}
\]
\[
\text{JACOB(POS,POS+NY) = C1FX}
\]

**GO TO 40**

**LEFT INTERIOR POINTS**

\[
\text{JACOB(POS,POS+1) = C2FX \cdot WP(FY)}
\]
\[
\text{JACOB(POS,POS-1) = C2FX \cdot WM(FY)}
\]
\[
\text{JACOB(POS,POS+NYM1) = C1FX}
\]
\[
\text{JACOB(POS,POS-NY) = C1FX}
\]

**GO TO 70**

**LOWER BOUNDARY POINTS**

\[
\text{JACOB(POS,POS+2) = DELW2M}
\]

**GO TO 70**

**UPPER BOUNDARY POINTS**

\[
\text{JACOB(POS,POS-2) = -DELW2M}
\]

\[
\text{PO} = \text{PO} + 1
\]
\[
\text{FY} = \text{FY} + 1
\]

**CONTINUE**

\[
\text{FX} = \text{FX} - 1
\]

**CONTINUE**

**LEFT BOUNDARY POINTS**

\[
\text{FX} = 1
\]
\[
\text{FY} = 2
\]
DO 100 IY = 1, NYM2
   JACOB(POS, POS-NYM1) = 0.5D0
   JACOB(POS, POS-NYNM1) = DELS2M
   FY = FY + 1
   POS = POS + 1
100 CONTINUE
C
RETURN
C
END
C
C SUBROUTINE FDIAG
C
C Defines diagonal elements of Jacobian array JACOB for solution of difference system by direct means.
C
Parameters:
C
JACOB(NXJ, NYJ) -> Jacobian array
PHI(NX, NY) -> Unknown function
KSQR(NX, NY) -> Extrinsic curvature squared
S2, S4 -> Radial coordinate functions
WP, WM -> Angular "
DELS -> Radial mesh spacing
DELM -> Angular "

C
C SUBROUTINE FDIAG(JACOB, NXJ, NYJ, PHI, KSQR, NX, NY, S2, S4, WP, WM, * DELS, DELW)
C
INTEGER NX, NXJ, NY, NYJ
REAL*8 JACOB(NXJ, NYJ), PHI(NX, NY), KSQR(NX, NY)
REAL*8 S2(1), S4(1), WP(1), WM(1)
REAL*8 CIFX, C2FX, DELS, DELS2M, DELSM1, DELSM2, DELW, DELW2M, DELWM1, DELWM2

INTEGER IX, IY, FX, FY, NXM1, NXM2, NYM1, NYM2

NXM1 = NX - 1
NXM2 = NX - 2
NYM1 = NY - 1
NYM2 = NY - 2
DELSM1 = 1.0D0 / DELS
DELSM2 = DELSM1 ** 2
DELS2M = 0.5D0 * DELSM1
DELWM1 = 1.0D0 / DELW
DELWM2 = DELWM1 ** 2
DELW2M = 0.5D0 * DELWM1

FX = NXM1
POS = 1

DO 50 IX = 1, NXM2
  FY = 1
  C1FX = DELSM2 * S4(FX)
  C2FX = DELWM2 * S2(FX)

DO 40 IY = 1, NY
  IF( IY .EQ. 1 ) GO TO 10
  IF( IY .EQ. NY ) GO TO 20

INTERIOR POINTS

JACOB(POS,POS) = -(2.0D0 * C1FX + C2FX * (WP(FY) + WM(FY)) + 0.875D0 * KSQR(FX,FY) /
  (PHI(FX,FY) ** 8))

GO TO 30

LOWER BOUNDARY POINTS

10 JACOB(POS,POS) = -DELW2M

GO TO 30

UPPER BOUNDARY POINTS

20 JACOB(POS,POS) = DELW2M

30 POS = POS + 1
  FY = FY + 1
40 CONTINUE

FX = FX - 1
50 CONTINUE

LEFT BOUNDARY POINTS

FX = 1
FY = 2

DO 60 IY = 1, NYM2
  JACOB(POS,POS) = -DELS2M
  FY = FY + 1
  POS = POS + 1
60 CONTINUE

RETURN

END

-----------------------------

SUBROUTINE DIRSPC
ALLOCS space for direct solution of coarse grid
difference equations and calls FOFFD to define (static)
off-diagonal Jacobian matrix elements.

Parameters:

MEMORY -> Main storage array
MEMPT --> Current available location in MEMORY
NX, NY --> Dimensions of coarse grid
S2, S4 --> Radial coordinate functions
WP, WM --> Angular "
DELS --> Radial mesh spacing
DELW --> Angular "

Routines called:

FOFFD

SUBROUTINE DIRSPC(MEMORY, MEMPT, NX, NY, S2, S4, WP, WM, DELS, DELW)

REAL*8 MEMORY(1), S2(1), S4(1), WP(1)
REAL*8 DELS, DELW
INTEGER DELPHI, JACOB, MEMPT, NJ, NX, NXW, NY, NYW, WORK
COMMON / DIR / JACOB, NJ, WORK, NXW, NYW, DELPHI

NJ = (NX - 2) * NY + (NY - 2)
NXW = NJ
NYW = NXW + 1
JACOB = MEMPT
MEMPT = MEMPT + NJ ** 2
WORK = MEMPT
MEMPT = MEMPT + NXW * NYW
DELPHI = MEMPT
MEMPT = MEMPT + NJ

DEFINE OFF-DIAGONAL JACOBIAN ELEMENTS
CALL FOFFD(MEMORY(JACOB), NJ, NX, NY, S2, S4, WP, WM, DELS, DELW)
RETURN
END
INTERPOLATION AND RESTRICTION

ROUTINES

SUBROUTINE INTERP

Calls either LININT to perform linear interpolation or CUBINT to do cubic interpolation of coarse grid function to fine grid. Coarse-to-fine x, y mesh spacing ratios are assumed to be either 2:1 or 1:1. Coarse grid domain is assumed to contain fine grid domain, and all interior coarse grid lines must be fine grid lines.

Parameters:

MEMORY -> Main storage array
CGRID --> Coarse grid number
CFPT ---> Pointer to start of coarse grid fcn in MEMORY
FGRID --> Fine grid number
FFPT ---> Pointer to start of fine grid fcn in MEMORY
ORDER --> .EQ. 4 -> Perform cubic interpolation
. NE. 4 -> " linear"

Routines called:

BIG
DFLOAT
SMALL
TRNCAT
CEIL
LININT
CUBINT

SUBROUTINE INTERP(MEMORY, CGRID, CFPT, FGRID, FFPT, ORDER, LIST, UNIT)

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)

INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
* SDN(50), NPT(50), NX(50), NY(50), XHIB(50),
* XLOB(50), YHIB(50), YLOB(50)

REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
* XINC(50), YINC(50)

LOGICAL CONV(50), SLOW(50)
INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
INTEGER MEMPT
INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC
INTEGER KSQR, R, RHs, RM, RP, TAU, W, WM, WP, U, OLDU
INTEGER NULL / 280808080 /
COMMON / LHEAD, LLINK, FATHER, BROTHER, SON,
NX, NY, NPT, AREA, XINC, YINC,
XSTART, XHIB, YSTART, YHIB, ERROR, CONV, SLOW, EPSI,
NRM, TAUS, CRSLEV, FINLEV, NGRID, MEMPT
COMMON / R, RP, RM, R, W, WP, WM, U, OLDU, RHs,
TAUS, CRSLEV, FINLEV, NYFUNC, NGFUNC

REAL*8 MEMORY(1)
REAL*8 BIG, DFLOAT, SMALL, TRNCAT
REAL*8 CFX1, CFX2, CFY1, CFY2, CXO, CXINC, CYO,
CYINC, FXO, FXINC, FYO, FYINC
INTEGER CEIL
INTEGER CFPT, CGRID, CIXFIN, CIYFIN, CIYST, CNX,
CNY, FFPT, FGRID, FIXFIN, FIXST, FYFIN, FYST,
FX, FY, FTXST, FTYST, LIST, NCX, NCY,
ORDER, TNX, TNY, UNIT, XBRST, XSTEP,
YBRST, YSTEP

CNX = NX(CGRID)
CNY = NY(CGRID)
FNX = NX(FGRID)
FNY = NY(FGRID)
CXINC = XINC(CGRID)
CYINC = YINC(CGRID)
FXINC = XINC(FGRID)
FYINC = YINC(FGRID)
CXO = MEMORY(XFSTART(CGRID,R))
CYO = MEMORY(YFSTART(CGRID,R))
FXO = MEMORY(XFSTART(FGRID,R))
FYO = MEMORY(YFSTART(FGRID,R))

DETERMINE RELATIONSHIP OF FINE GRID BOUNDARY W.R.T
COARSE GRID BOUNDARY

CFX1 = (FXO - CXO) / CXINC
CFY1 = (FYO - CYO) / CYINC
CFX2 = (FXO + (FNX - 1) * FXINC - CXO) / CXINC
CFY2 = (FYO + (FNY - 1) * FYINC - CYO) / CYINC
CIYFIN = SMALL(DFLOAT(CNX),DFLOAT(CEIL(CFX2)) + 1.000)
CIYFIN = SMALL(DFLOAT(CNX),DFLOAT(CEIL(CFY2)) + 1.000)
NCX = CIXFIN - CIXST + 1
NCY = CIYFIN - CIYST + 1

C IF( CXINC .EQ. FXINC ) GO TO 10
   XSTEP = 1
   GO TO 20
10   XSTEP = 2
20   IF( CYINC .EQ. FYINC ) GO TO 30
   YSTEP = 1
   GO TO 40
30   YSTEP = 2
40   IF( ORDER .EQ. 4 .AND. NCX .GT. 3 .AND. NCY .GT. 3 )
      GO TO 70

C LINEAR INTERPOLATION
C IF( CFX1 .LT. 0.000 ) GO TO 50
   FIXST = 1
   FIXFIN = FNX
   XBRST = 0
   IF( CFX1 .NE. TRNCAT(CFX1) ) XBRST = 1
   GO TO 60
50   FIXST = 2
   FIXFIN = FNX
   XBRST = 0
60   FIYST = 1
   FIYFIN = FNY
   YBRST = 0
   IF( CFY1 .NE. TRNCAT(CFY1) ) YBRST = 1
   CALL LININT(MEMORY(CFPT),MEMORY(FFPT),CNX,CNY,
               FNX,FNY,CIXST,CIYST,FIXST,FIXFIN,
               FIYST,FIYFIN,XBRST,YBRST,XSTEP,YSTEP)

C GO TO 140
C CUBIC INTERPOLATION
C 70 IF( XSTEP .EQ. 2 ) GO TO 80
   TNX = 2 * NCX - 1
   GO TO 90
80  TNX = CNX
90 IF( YSTEP .EQ. 2 ) GO TO 100
   TNY = 2 * NCY - 1
   GO TO 110
100  TNY = CNY
110 IF( CFX1 .LT. 0.000 ) GO TO 120
   FIXST = 1
   FIXFIN = FNX
   FTXST = 1
   IF( CFX1 .NE. TRNCAT(CFX1) ) FTXST = 2
   GO TO 130
120  FIXST = 2
130  FLYST = 1
     FLYFIN = FNY
     FTYST = 1
IF( CFY1 .NE. TRNCAT(CFY1) ) FTYST = 2

CALL CUBINT(MEMORY(CFPT),MEMORY(FFPT),MEMORY(MEMPT),
CNX,CNY,FNX,FNY,TNX,TNY,CIXST,CIXFIN,
* CIYST,CIYFIN,FIXST,FIXFIN,FIYST,FIYFIN,
* FTXST,FTYST,XSTEP,YSTEP)

RETURN

END

SUBROUTINE LININT

Performs linear interpolation of coarse grid function
COARSE to fine grid function FINE. See subroutine INTERP
for assumptions regarding grid spacings etc.

Parameters:

COARSE(CNX,CNY) -> Coarse grid function
FINE(FNX,FNY) ----> Fine grid function
CIXST, CIYST ----> Starting indices for interpolation
FIXST, FIYST ----> Indices for first interpolated value
FIXFIN, FIYFIN ----> Indices for last interpolated value
XBRST, YBRST ----> Flags indicating positioning of fine
grid w.r.t coarse grid
XSTEP, YSTEP ----> Coarse-to-fine mesh spacing ratios

Routines called:

MOD

SUBROUTINE LININT(COARSE,FINE,CNX,CNY,FNX,FNY,CIXST,
* CIYST,FIXST,FIXFIN,FIYST,FIYFIN,
* XBRST,YBRST,XSTEP,YSTEP)

INTEGER MOD
INTEGER CNX, CNY, FNX, FNY
REAL*8 COARSE(CNX,CNY), FINE(FNX,FNY)
INTEGER CIX, CIXST, CIY, CIYST, FIX, FIXFIN,
* FIXST, FIY, FIYFIN, FIYST, XBR,
* XBRST, XSTEP, YBR, YBRST, YSTEP

CIY = CIYST
YBR = YBRST
SUBROUTINE CUBINT

Performs cubic interpolation of coarse grid function COARSE to fine grid function FINE. See subroutine INTERP for assumptions regarding grid spacings etc. Interpolation is first performed into array TEMP which corresponds to a fine grid which covers the same region as the coarse grid. Appropriate values are then transferred to FINE.

Parameters:

- COARSE(CNX,CNY) -> Coarse grid function
- FINE(FNX,FNY) ---> Fine grid function
- TEMP(TNX,TNY) ---> Temporary work array
- CIXST, CIYST ---> Starting indices for interpolation
- FIXST, FIYST ---> Indices for first interpolated value
- FIXFIN, FIYFIN ---> Indices for last interpolated value
- FTXST, FTYST ---> Starting indices for transfer of interpolated values from TEMP to FINE
- XSTEP, YSTEP ---> Coarse-to-fine mesh spacing ratios

Routines called:
SUBROUTINE CUBINT(COARSE,FINE,TEMP,CNX,CNY,FNX,FNY,
  *  TNX,TNY,CIXST,CIXFIN,CIYST,CIYFIN,
  *  FIXST,FIXFIN,FIYST,FIYFIN,FTXST,
  *  FTYST,XSTEP,YSTEP)

INTEGER CNX, CNY, FNX, FNY, TNX, TNY
REAL*8 COARSE(CNX,CNY),  FINE(FNX,FNY), TEMP(TNX,TNY)
REAL*8 CUBIC1
INTEGER MOD

INTEGER CIX, CIXFIN, CIXST, CIY, CIYFIN, CIYST,
  *  CXINT, FIX, FIXFIN, FIXST, FIY, FIYFIN, FIYST,
  *  FTXST, FTYST, IINT, NXINT, NXINT1,
  *  NYINT, NYINT1, T1, T2, TIX, TIY, TYINT,
  *  XBR, XSTEP, YBR, YSTEP

T1 = CIXFIN - CIXST
T2 = CIYFIN - CIYST
NXINT = 0
NYINT = 0
IF( XSTEP .EQ. 1 ) NXINT = T1
IF( YSTEP .EQ. 1 ) NYINT = T2
NXINT1 = NXINT - 1
NYINT1 = NYINT - 1

INTERPOLATE IN X DIRECTION

TIY = 1
DO 40 CIY = CIYST, CIYFIN
  CIX = CIXST
  CXINT = CIXST
  XBR = 0
  IINT = 1
  DO 30 TIX = 1, TNX
    IF( XBR .EQ. 1 ) GO TO 10
    TEMP(TIX, TIY) = COARSE(CIX, CIY)
    CIX = CIX + 1
    GO TO 20
  10  TEMP(TIX, TIY) = CUBIC1(COARSE, CNX, CNY, CXINT, CIX, CIY, IINT, NXINT)
    IF( IINT .LE. 1 OR .INT .GE. NXINT1 ) GO TO 15
    CXINT = CXINT + 1
    IINT = IINT + 1
    XBR = MOD(XBR + XSTEP, 2)
  15  CONTINUE
  20  CONTINUE
  30  CONTINUE
  40  CONTINUE
CONTINUE

IF( YSTEP .EQ. 2 ) GO TO 80

INTERPOLATE IN Y DIRECTION

DO 70 TIX = 1, TNX
  YBR = 0
  TYINT = 1
  IINT = 1

DO 60 TIY = 1, TNY
  IF( YBR .EO. 0 ) GO TO 50
  TEMP(TIX, TIY) = CUBIC1(TEMP, TNX, TNY, TIX, TYINT, 1, IINT, NYINT)
  IF( IINT .LE. 1 .OR. IINT .GE. NYINT )
    GO TO 45
  TYINT = TYINT + 2
  IINT = IINT + 1
45  IINT = IINT + 1
50  YBR = MOD(YBR + YSTEP, 2)
60  CONTINUE

CONTINUE

80  TIY = FTYST

TRANSFER VALUES FROM TEMP TO FINE

DO 100 FIY = FIYST, FIYFIN
  TIX = FTXST

DO 90 FIX = FIXST, FIXFIN
  FINE(FIX, FIY) = TEMP(TIX, TIY)
  TIX = TIX + 1
90  CONTINUE

TIY = TIY + 1

CONTINUE

RETURN

END

FUNCTION CUBIC1

Helper routine for SUBROUTINE CUBINT. Determines which
direction interpolation is being performed in, calls CUBIC
to produce interpolated value.

Parameters:

F(NX,NY) --> Function values used for interpolation
I, J --> Indices of interpolation point (IP)
RDWCOL --> .EQ. 0 --> Interpolation in x direction
DOUBLE PRECISION FUNCTION CUBIC1(F,NX,NY,I,J,ROWCOL,IINT,NINT)

INTEGER NX, NY
REAL*8 F(NX,NY)
REAL*8 CUBIC
REAL*8 FO, F1, F2, F3, FIVE, ONE, THREE
INTEGER I, IINT, J, NINT, ROWCOL
DATA ONE / 1.0DO /, THREE / 3.0DO /, FIVE / 5.0DO /

IF( ROWCOL .EQ. 1 ) GO TO 10
FO = F(I,J)
F1 = F(I+1,J)
F2 = F(I+2,J)
F3 = F(I+3,J)
GO TO 20
10
FO = F(I,J)
F1 = F(I,J+2)
F2 = F(I,J+4)
F3 = F(I,J+6)

IF( IINT .EQ. 1 ) GO TO 40
IF( IINT .EQ. NINT ) GO TO 30
CUBIC1 = CUBIC(THREE,FO,F1,F2,F3)
GO TO 50
CUBIC1 = CUBIC(FIVE,FO,F1,F2,F3)
GO TO 50
40
CUBIC1 = CUBIC(ONE,FO,F1,F2,F3)
50
RETURN

END

FUNCTION CUBIC

Helper routine for FUNCTION CUBIC1. Returns interpolated value from FO, F1, F2, F3. N (1,3,5) specifies positioning
of interpolated value with respect to \( F_0 \).

Parameters: As above

---

DOUBLE PRECISION FUNCTION CUBIC(N, \( F_0 \), \( F_1 \), \( F_2 \), \( F_3 \))

```
REAL*8 \( F_0 \), \( F_1 \), \( F_2 \), \( F_3 \), N, T1, T2, T3

T1 = N * 0.5D0
T2 = T1 * ( N - 2.0D0 ) * 0.25D0
T3 = T2 * ( N - 4.0D0 ) / 6.0D0
CUBIC = \( F_0 \) + T1 * ( \( F_1 \) - \( F_0 \) ) +
        T2 * ( \( F_2 \) - 2.0D0 * \( F_1 \) + \( F_0 \) ) +
        T3 * ( \( F_3 \) - 3.0D0 * \( F_2 \) + 3.0D0 * \( F_1 \) - \( F_0 \) )

RETURN
```

---

SUBROUTINE INJECT

Restricts fine grid function \( FINE \) to coarse grid function \( COARSE \) by injecting appropriate interior values and possibly extrapolating boundary values. See routine INTERP for assumed relation between coarse and fine grids.

Parameters:

\( COARSE(CNX, CNY) \rightarrow \) Coarse grid (CG) function

\( FINE(FNX, FNY) \rightarrow \) Fine grid (FG) function

\( CXO, CYO \rightarrow \) Coords. of lower left corner of CG

\( FXO, FYO \rightarrow \) FG

\( CXINC, CYINC \rightarrow \) x,y mesh spacings of CG

\( FXINC, FYINC \rightarrow \) FG

\( CXLOB, CYHIB, \rightarrow \) CG boundary flags

\( CYLOB, CYHIB, \rightarrow \) FG boundary flags

\( MODE \rightarrow \) .EQ. 0 \rightarrow extrapolate boundary values

\( .NE. 0 \rightarrow \) inject boundary values

Routines called:

\( CEIL \)

\( TRNCAT \)

---

SUBROUTINE INJECT(COARSE, FINE, CNX, CNY, FNX, FNY, CXO, CYO, ...)
* CYO,FXO,FYO,CXINC,CYINC FXINC,FYINC,
* CXLOB,CXHIB,CYLOB,CYHIB,FXLOB,FXHIB,
* FYLOB,FYHIB,MODE)
C
INTEGER CNX, CNY, FNX, FNY
C
REAL*8 COARSE(CNX,CNY), FINE(FNX,FNY)
C
REAL*8 TRNCAT
C
REAL*8 CFX, CFY, CXO, CXINC, CYO, CYINC, FXO,
FXINC, FYO, FYINC
C
INTEGER CEIL
C
INTEGER CXHIB, CXLOB, CYHIB, CYLOB, FXHIB, FXLOB,
FYHIB, FYLOB
C
INTEGER CIX, CIXFIN, CIXST, CIY, CIYFIN, CIYST,
* FIX, FIXST, FIY, FIYST,
* FNXM1, FNXM2, FNXM3, FNYM1, FNYM2, FNYM3,
* MODE, NIX, NIY, XSTEP, YSTEP
C
CFX = (FXO - CXO) / CXINC
CFY = (FYO - CYO) / CYINC
IF(CFX .NE. TRNCAT(CFX) ) GO TO 10
CIXST = CFX + 1.0DO
FIXST = 1
GO TO 20
CIXST = CFX + 1.5D0
FIXST = 2
20 IF(CFY .NE. TRNCAT(CFY) ) GO TO 30
CIYST = CFY + 1.0DO
FIYST = 1
GO TO 40
30 CIYST = CFY + 1.5DO
FIYST = 2
40 NIX = CEIL(((FNX + 1 - FIXST) * FXINC) / CXINC)
NIY = CEIL(((FNY + 1 - FIYST) * FYINC) / CYINC)
CIXFIN = CIXST + NIX - 1
CIYFIN = CIYST + NIY - 1
IF( FXINC .EQ. CXINC ) GO TO 50
XSTEP = 2
GO TO 60
XSTEP = 1
50 IF( FYINC .EQ. CYINC ) GO TO 70
YSTEP = 2
GO TO 80
YSTEP = 1
60 FIX = FIXST
70 FIX = FIXST
80
C
INTERIOR INJECTION
C
DO 100 CIX = CIXST, CIXFIN
FIY = FIYST
C
DO 90 CIY = CIYST, CIYFIN
COARSE(CIX,CIY) = FINE(FIX,FIY)
FIY = FIY + YSTEP
CONTINUE
C
FIX = FIX + XSTEP
CONTINUE
C
BOUNDARY EXTRAPOLATION
C
IF( ( CYHIB .EQ. 0 .AND. FYHIB .EQ. 0 ) .OR. YSTEP .EQ. 1 ) GO TO 120
C
EXTRAPOLATE "UPPER" BOUNDARY
C
FIX = FIXST
FNYM1 = FNY - 1
FNYM2 = FNY - 2
FNYM3 = FNY - 3
IF( MODE .NE. 0 ) GO TO 112
C
DO 110 CIX = CIXST , CIXFIN
COARSE(CIX,CNY) = 3.5D0 * FINE(FIX,FNY) -
*        4.5D0 * FINE(FIX,FNYM1) +
*        2.5D0 * FINE(FIX,FNYM2) -
*        0.5D0 * FINE(FIX,FNYM3)
C
CONTINUE
110 CONTINUE
GO TO 120
C
112 DO 115 CIX = CIXST , CIXFIN
COARSE(CIX,CNY) = FINE(FIX,FNY)
C
CONTINUE
115 CONTINUE
C
IF( ( CYLOB .EQ. 0 .AND. FYLOB .EQ. 0 ) .OR. YSTEP .EQ. 1 ) GO TO 140
C
EXTRAPOLATE "LOWER" BOUNDARY
C
FIX = FIXST
IF( MODE .NE. 0 ) GO TO 132
C
DO 130 CIX = CIXST , CIXFIN
COARSE(CIX,1) = 3.5D0 * FINE(FIX,1) -
*        4.5D0 * FINE(FIX,2) +
*        2.5D0 * FINE(FIX,3) -
*        0.5D0 * FINE(FIX,4)
C
CONTINUE
130 CONTINUE
GO TO 140
C
132 DO 135 CIX = CIXST , CIXFIN
COARSE(CIX,1) = FINE(FIX,1)
C
CONTINUE
135 CONTINUE
C
IF( ( CXHIB .EQ. 0 .AND. FXHIB .EQ. 0 ) .OR. XSTEP .EQ. 1 ) GO TO 160
C
EXTRAPOLATE "OUTER" BOUNDARY
C
FIY = FIYST
FNXM1 = FNX - 1
FNXM2 = FNX - 2
FNXM3 = FNX - 3
IF( MODE .NE. 0 ) GO TO 152

C
DO 150 CIY = CIYST , CIYFIN
   COARSE(CNX,CIY) = 3.5D0 * FINE(FNX,FIY) - 
   4.5D0 * FINE(FNXM1,FIY) + 
   2.5D0 * FINE(FNXM2,FIY) - 
   0.5D0 * FINE(FNXM3,FIY)
   FIY = FIY + YSTEP
 150 CONTINUE
GO TO 160
C
152 DO 155 CIY = CIYST , CIYFIN
   COARSE(CNX,CIY) = FINE(FNX,FIY)
   FIY = FIY + YSTEP
 155 CONTINUE
C
160 IF( ( CXLOB .EQ. 0 .AND. FXLOB .EQ. 0 ) .OR. 
   XSTEP .EQ. 1 ) GO TO 180
C
C EXTRAPOLATE "INNER" BOUNDARY
C
FIY = FIYST
IF( MODE .NE. 0 ) GO TO 172
C
DO 170 CIY = CIYST , CIYFIN
   COARSE(1,CIY) = 3.5D0 * FINE(1,FIY) - 
   4.5D0 * FINE(2,FIY) + 
   2.5D0 * FINE(3,FIY) - 
   0.5D0 * FINE(4,FIY)
   FIY = FIY + YSTEP
 170 CONTINUE
GO TO 180
C
172 DO 175 CIY = CIYST , CIYFIN
   COARSE(1,CIY) = FINE(1,FIY)
   FIY = FIY + YSTEP
 175 CONTINUE
C
180 RETURN
C
END
ROUTINES FOR GRID DEFINITION
AND INITIALIZATION: FUNCTION DEFINITION

SUBROUTINE INIGST

Initializes grid structure

Parameters:

INILEV -> Initial level (usually 1 if first level to be used will be coarsest level throughout run)
MAXGRD -> Maximum number of grids to be used in run

SUBROUTINE INIGST(INILEV, MAXGRD)

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)

INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
SON(50), NPT(50), NX(50), NY(50), XHIB(50),
XLOB(50), YHIB(50), YLOB(50)

REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
XINC(50), YINC(50)

LOGICAL CONV(50), SLOW(50)

INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC

INTEGER MEMPT

INTEGER KSQR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU

INTEGER NULL / ZB0B0B0B0 /

COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHR, SON,
NX, NY, NPT, AREA, XINC, YINC,
XFSTRT, YFSTRT, GFSTRT, XLOB, XHIB,
YLOB, YHIB, ERROR, CONV, SLOW, EPSI,
NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT

COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDU, RHS,
TAU, KSQR, NXFUNC, NYFUNC, NGFUNC
INTEGER I, INILEV, MAXGRD

INITIALIZE TREE STRUCTURE

DO 10 I = 1, MAXGRD
  LHEAD(I) = NULL
  LLINK(I) = NULL
  FATHER(I) = NULL
  BROTHER(I) = NULL
  SON(I) = NULL
10 CONTINUE

INITIALIZE FUNCTION "POINTERS"

MEMPT = 1
NGRID = 0
CRSLEV = INILEV
FINLEV = INILEV
NXFUNC = 3
NYFUNC = 3
NGFUNC = 5
R = 1
RP = 2
RM = 3
W = 1
WP = 2
WM = 3
U = 1
OLDU = 2
RHS = 3
TAU = 4
KSQR = 5

RETURN

SUBROUTINE NEWGRD

Attempts to insert new null grid into grid structure. Grid is at level LEV and has father DAD. MAXGRD is the maximum number of grids which may be defined. RC returns with a non-zero value if insertion attempt is unsuccessful.

Parameters: As above

SUBROUTINE NEWGRD(LEV, DAD, MAXGRD, RC)
GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)

INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
      SDN(50), NPT(50), NX(50), NY(50), XHIB(50),
      XLOB(50), YHIB(50), YLOB(50)

REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
      XINC(50), YINC(50)

LOGICAL CONV(50), SLOW(50)

INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC

INTEGER MEMPT

INTEGER CRSLEV1, DAD, FINLV1, LEV, MAXGRD, NGRID1, PSON, RC

NGRID1 = NGRID + 1
IF( NGRID1 .GT. MAXGRD ) GO TO 60

DO GRIDS ALREADY EXIST AT THIS LEVEL ?

IF( LHEAD(LEV) .EQ. NULL ) GO TO 10

IF SO, THEN A FATHER MUST BE SPECIFIED

IF( DAD .EQ. NULL ) GO TO 30

INSERT NEW GRID INTO STRUCTURE, UPDATE Pointers

PSON = SDN(DAD)
SON(DAD) = NGRID1
FATHER(NGRID1) = DAD
BROTHR(NGRID1) = PSON
LLINK(NGRID1) = LHEAD(LEV)
LHEAD(LEV) = NGRID1

GO TO 20

CHECK FOR INVALID LEVEL SPECIFICATION

CRSLV1 = CRSLEV - 1
FINLV1 = FINLEV + 1
IF( LEV .LT. CRSLV1 ) GO TO 40
IF( LEV .EQ. CRSLV1 ) CRSLEV = CRSLV1
IF( LEV .GT. FINLV1 ) GO TO 50
IF( LEV .EQ. FINLV1 ) FINLEV = FINLV1

INSERT NEW GRID, UPDATE POINTERS

LHEAD(LEV) = NGRID1
LLINK(NGRID1) = NULL
FATHER(NGRID1) = DAD
IF( DAD .NE. NULL ) SON(DAD) = NGRID1
SON(NGRID1) = LHEAD(LEV + 1)
BROTHR(NGRID1) = NULL

C

20 NGRID = NGRID1
C
NORMAL RETURN
RC = 0
RETURN

C

ATTEMPT TO INSERT GRID AT EXISTING LEVEL WITH NULL FATHER
30 RC = 1
RETURN

C

SPECIFIED INSERTION LEVEL TOO COARSE
40 RC = 2
RETURN

C

SPECIFIED INSERTION LEVEL TOO FINE
50 RC = 3
RETURN

C

NO MORE SPACE AVAILABLE IN GRID STRUCTURE
60 RC = 4
RETURN
C
END

-----------------------------------------------

SUBROUTINE INIGRD

-----------------------------------------------

Initializes coordinate and grid functions.

Parameters:

KSOR(NS,NW) --> Extrinsic curvature squared
RHS(NS,NW)  --> Right-hand-side of grid equations
S, S2, S4  --> Radial coordinate functions
W, WP, WM  --> Angular
SUBROUTINE INIGRD(KSQR,RHS,S,S2,S4,W,WP,WM,NS,NW,SMNGRD, * WMNGRD,DELS,DELW,A,P)

INTEGER NS, NW
REAL*8 KSQR(NS,NW), RHS(NS,NW)
REAL*8 S(1), S2(1), S4(1), W(1), WM(1), WP(1)
REAL*8 DSIN
REAL*8 A, DELS, DELW, DELW2, DSWJM1, P, SI, SIM1, SMNGRD,
Wd, WMNGRD
INTEGER I, J, NSM1

DELW2 = 0.5DO * DELW
S(1) = SMNGRD
W(1) = -DELW
W(2) = O.ODO
WP(2) = 2.ODO
WM(2) = 2.ODO

DO 10 I = 2 , NS
   SI = S(I-1) + DELS
   S(I) = SI
   SIM1 = 1.ODO - SI
   S2(I) = SIM1 ** 2
   S4(I) = SIM1 ** 4
CONTINUE

10 DO 20 J = 3 , NW
   WJ = W(J-1) + DELW
   DSWJM1 = 1.ODO / DSIN(WJ)
   W(J) = WJ
   WP(J) = DSWJM1 * DSIN(WJ + DELW2)
   WM(J) = DSWJM1 * DSIN(WJ - DELW2)

CONTINUE

CALL INIT(RHS,NS,NW,O.ODO)
CALL DEFKSQ(KSQR,NS,NW,S,W,A,P)
SUBROUTINE DEFGRD

Initializes scalar grid quantities such as area, mesh spacings, convergence criterion, etc. Determines boundary types and initializes boundary flags. Allocates space for grid and coordinate functions.

Parameters:
- GRID  > Grid number
- XMNGRD, XMXGRD --> x coords of grid boundaries
- YMNGRD, YMXGRD --> y " " " " domain "
- XMNDOM, XMXDOM --> x " " domain "
- YMNDOM, YMXDOM --> y " " " "
- NWXINC, NWYINC --> x,y mesh spacings
- NXLO, NXHI, ----> Boundary flags
- NYLD, NYHI /
- EPSILN --------> Initial convergence criterion
- NWAREA --------> Area of grid

Routines called:
- DABS

GRID DATA STRUCTURE

INTEGER GFSTRT(50,10), XFSTRT(50,10), YFSTRT(50,10)

INTEGER BROTHR(50), FATHER(50), LHEAD(50), LLINK(50),
SDN(50), NPT(50), NX(50), NY(50), XHIB(50),
XL0B(50), YHIB(50), YL0B(50)

REAL*8 AREA(50), EPSI(50), ERROR(50), NRMTAU(50),
XINC(50), YINC(50)

LOGICAL CONV(50), SLOW(50)

INTEGER CRSLEV, FINLEV, NGFUNC, NGRID, NXFUNC, NYFUNC

INTEGER MEMPT

INTEGER KSOR, R, RHS, RM, RP, TAU, W, WM, WP, U, OLDU

RETURN
END
INTEGER NULL / Z80808080 / 

COMMON / COMGST / LHEAD, LLINK, FATHER, BROTHER, SON, 
* 
NX, NY, NPT, AREA, XINC, YINC, 
* 
XFSRTRT, YFSRTRT, GFSRT, XLOB, XHIB, 
* 
YLOB, YHIB, ERROR, CONV, SLOW, EPSI, 
* 
NRMTAU, CRSLEV, FINLEV, NGRID, MEMPT 

COMMON / COMCON / R, RP, RM, W, WP, WM, U, OLDMU, RHS, 
* 
TAU, KSQR, NFXUNC, NFXUNC, NGFUNC 

REAL*8 

REAL*8 

REAL*8 

INTEGER 

DELXMN, DELXMX, DELYMN, DELYM, EPSILN, NWXINC, 
* 
NYINC, XL, XMNDOM, XMNGRD, XMXDOM, XMXGRD, YL, 
* 
YMNDOM, YMNGRD, YMXDOM, XMXGRD, NWAREA 

REAL*8 

PIBY2 / Z411921FB54442D18 / 

INTEGER IGFUNC, IXXFUN, IYYFUN, GRID, NPOINT, NXGRID, 
* 
NXH1, NXLO, NYGRID, NYHI, NYLO 

DELXMN = XMNGRD - XMNDOM 
DELXMX = XMXDOM - XMNGRD 
DELYMN = YMNGRD - YMNDOM 
DELYMX = YMXDOM - YMXGRD 

ESTABLISH THE 4 BOUNDARY TYPES 

DIRICHLET: 0, NEUMANN, ROBBIN: > 0 

IF( DELXMN .GE. 0.000) GO TO 10 
    XLOB(GRID) = NXLO 
    IF( DABS(DELXMN) .GT. NWXINC) XMNGRD = XMNGRD + NWXINC 
GO TO 20 
10    XLOB(GRID) = 0 
20    IF( DELXMX .GE. 0.000) GO TO 30 
    XHIB(GRID) = NXHI 
    IF( DABS(DELXMX) .GT. NWXINC) XMXGRD = XMXGRD - NWXINC 
GO TO 40 
30    XHIB(GRID) = 0 
40    IF( DELYMN .GE. 0.000) GO TO 50 
    YLOB(GRID) = NYLO 
    IF( DABS(DELYMN) .GT. NWYINC) YMNGRD = YMNGRD + NWYINC 
GO TO 60 
50    YLOB(GRID) = 0 
60    IF( DELYMX .GE. 0.000) GO TO 70 
    YHIB(GRID) = NYHI 
    IF( DABS(DELYMX) .GT. NWYINC) YMXGRD = YMXGRD - NWYINC 
GO TO 80 
70    YHIB(GRID) = 0 
80    XL = XMXGRD - XMNGRD 
    YL = YMXGRD - YMNGRD 
    NXGRID = XL / NWXINC + 1 
    NYGRID = YL / NWYINC + 1 
    XINC(GRID) = NWXINC
YINC(GRID) = PIBY2 / (NYGRID - 3)
NX(GRID) = NXGRID
NY(GRID) = NYGRID
NPOINT = NXGRID * NYGRID
NPT(GRID) = NPOINT
AREA(GRID) = NWAREA
EPSI(GRID) = EPSILN

C ALLOCATE SPACE FOR COORDINATE AND GRID FUNCTIONS
C
DO 90 IXFUNC = 1 , NXFUNC
   XFSTRT(GRID,IXFUNC) = MEMPT
   MEMPT = MEMPT + NXGRID
90 CONTINUE
C
DO 100 IYFUNC = 1 , NYFUNC
   YFSTRT(GRID,IYFUNC) = MEMPT
   MEMPT = MEMPT + NYGRID
100 CONTINUE
C
DO 110 IGFUNC = 1 , NGFUNC
   GFSTRT(GRID,IGFUNC) = MEMPT
   MEMPT = MEMPT + NPOINT
110 CONTINUE
C
RETURN
C
END

C------------------------------------------------------------
C
C SUBROUTINE GUESS
C------------------------------------------------------------
C
Calculates initial estimate of function PHI(NS,NW).
Radial coordinate array S is supplied as parameter.
Currently uniformly initializes PHI to 1.000

Parameters: As above

C------------------------------------------------------------
C
SUBROUTINE GUESS(PHI,S,NS,NW)
C
INTEGER NS, NW
REAL*8 PHI(NS,NW)
REAL*8 S(1)
INTEGER IS, JW

DO 20 IS = 1 , NS
DO 10 JW = 1 , NW  
PHI(IS,JW) = 1.0DO  
CONTINUE
C
DO 20 IS = 1 , NSM1  
RIS = 1.0DO / (1.0DO - S(IS))  
ARIS2 = (A / RIS) ** 2  
T1 = (1.0DO - ARIS2) ** 2  
KSQR(IS,JW) = P26 * T1 / RIS ** 4  
CONTINUE
C
CONTINUE
C
RETURN
C
END
C
---
C
SUBROUTINE DEFKSO
C
Defines "extrinsic curvature squared" (H) for exactly soluble model problem.
C
Parameters:
C
KSQR(NS,NW) -> Extrinsic curvature squared  
S ------------> Radial coordinates  
W ------------> Angular coordinates  
A ------------> Inversion radius  
P -------------> Linear momentum
C
SUBROUTINE DEFKSO(KSQR,NS,NW,S,W,A,P)
C
INTEGER NS, NW
REAL*8 KSQR(NS,NW)
REAL*8 S(1), W(1)
REAL*8 A, ARIS2, P, P26, RIS, T1, T2, WJW2
INTEGER IS, JW, NSM1
C
NSM1 = NS - 1
P26 = 6.0DO * P ** 2
C
DO 20 JW = 1 , NW  
DO 10 IS = 1 , NSM1  
RIS = 1.0DO / (1.0DO - S(IS))  
ARIS2 = (A / RIS) ** 2  
T1 = (1.0DO - ARIS2) ** 2  
KSQR(IS,JW) = P26 * T1 / RIS ** 4  
CONTINUE
C
KSQR(NS,JW) = P26
SUBROUTINE DEFKSQ

Defines "extrinsic curvature squared" (H) for boosted black holes.

Parameters:

- KSQR(NS,NW) -> Extrinsic curvature squared
- S ------------> Radial coordinates
- W ----------> Angular coordinates
- A -----------> Inversion radius
- P ------------> Linear momentum

Routines called:

DCOS

SUBROUTINE DEFKSQ(KSQR,NS,NW,S,W,A,P)

INTEGER NS, NW
REAL*8 KSQR(NS,NW)
REAL*8 S(1), W(1)
REAL*8 DCOS
REAL*8 A, ARIS2, P, P245, RIS, T1, T2, WJW2

INTEGER IS, JW, NSM1

NSM1 = NS - 1
P245 = 4.5DO * P ** 2

DO 20 JW = 1, NW
   WJW2 = DCOS(W(JW)) ** 2

DO 10 IS = 1, NSM1
   RIS = 1.0DO / (1.0DO - S(IS))
   ARIS2 = (A / RIS) ** 2
   T1 = 1.0DO * (-4.0DO + ARIS2) * ARIS2
   T2 = (1.0DO + ARIS2) ** 2 + 2.0DO * WJW2 * T1
   KSQR(IS,JW) = P245 * T2 / RIS ** 4
SUBROUTINE DEFKSO

Defines "extrinsic curvature squared" (H) for spinning black holes.

Parameters:

- KSQR(NS,NW) -> Extrinsic curvature squared
- S  -> Radial coordinates
- W  -> Angular coordinates
- A  -> Inversion radius (unused)
- J  -> Angular momentum

Routines called:
- DSIN

SUBROUTINE DEFKSO(KSQR,NS,NW,S,W,A,J)

INTEGER NS, NW
REAL*8 KSQR(NS,NW)
REAL*8 S(1), W(1)
REAL*8 DSIN
REAL*8 A, J, J18, J18WJ, RISM6, WJW2M1
INTEGER IS, JW, NSM1

NSM1 = NS - 1
J18 = 18.0D0 * J ** 2

DO 20 JW = 1 , NW
    WJW2M1 = DSIN(W(JW)) ** 2
    J18WJ = J18 * WJW2M1
    DO 10 IS = 1 , NSM1

CONTINUE
KSQR(NS,JW) = P245 * (1.0D0 + 2.0D0 * WJW2)

CONTINUE
RETURN

END
RISMG = (1.00D0 - S(IS)) ** 6
KSQR(IS,JW) = J18WJ * RISMG
CONTINUE

KSQR(NS,JW) = 0.00D0
CONTINUE
RETURN

END
UTILITY ROUTINES FOR ARRAY OPERATIONS

SUBROUTINE ADDSUB

Adds (subtracts) entries of F2 to (from) F1, placing results in FRES

Parameters:

F1(FNX,FNY) --> As above
F2(FNX,FNY) /
FRES(FNX,FNY) /
SIGN --------------> .EQ. +1.0D0 --> Add
               .EQ. -1.0D0 --> Subtract

SUBROUTINE ADDSUB(F1,F2,FRES,FNX,FNY,SIGN)

INTEGER FNX, FNY
REAL*8 F1(FNX,FNY), F2(FNX,FNY), FRES(FNX,FNY)
REAL*8 SIGN
INTEGER IX, IY

DO 20 IX = 1, FNX
    DO 10 IY = 1, FNY
        FRES(IX,IY) = F1(IX,IY) + SIGN * F2(IX,IY)
    CONTINUE
20 CONTINUE

RETURN

END

SUBROUTINE COPY
C Copies array FROM(NX,NY) to array TO(NX,NY).
Parameters: As above

SUBROUTINE COPY(FROM,TO,NX,NY)
INTEGER NX, NY
REAL*8 FROM(NX,NY), TO(NX,NY)
INTEGER IX, IY

DO 20 IY = 1 , NY
   DO 10 IX = 1 , NX
      TO(IX,IY) = FROM(IX,IY)
   10 CONTINUE
20 CONTINUE
RETURN
END

SUBROUTINE INIT(ARRAY,NX,NY,VALUE)
INTEGER NX, NY
REAL*8 ARRAY(NX,NY)
REAL*8 VALUE
INTEGER IX, IY

DO 20 IY = 1 , NY
   DO 10 IX = 1 , NX
      ARRAY(IX,IY) = VALUE
   10 CONTINUE
20 CONTINUE
RETURN
SUBROUTINE VALURC

Places VALUE in all entries of a row or column of ARRAY(NX, NY)

Parameters: As above and

IRC --------> Row/column index
ROWCOL -----> .EQ. 0 -> Fill column
              .NE. 0 -> Fill row

SUBROUTINE VALURC(ARRAY, NX, NY, IRC, ROWCOL, VALUE)

INTEGER NX, NY
REAL*8 ARRAY(NX, NY)
REAL*8 VALUE
INTEGER IRC, IX, IY, ROWCOL

IF( ROWCOL .NE. 0 ) GO TO 20
   DO 10 IY = 1, NY
      ARRAY(IRC, IY) = VALUE
      CONTINUE
   10 GO TO 40

20   DO 30 IX = 1, NX
      ARRAY(IX, IRC) = VALUE
      CONTINUE
   30 RETURN

END
INPUT / OUTPUT ROUTINES

SUBROUTINE INDATA

Reads X(NX), Y(NY), KSQR(NX,NY), PHI(NX,NY) unformatted from logical unit FILE.

Parameters: As above

SUBROUTINE INDATA(X,Y,KSQR,PHI,NX,NY,FILE)

INTEGER NX, NY
REAL*8 KSQR(NX,NY), PHI(NX,NY), X(NX), Y(NY)
INTEGER IX, IY, FILE

READ(FILE) (X(IX), IX = 1, NX)
READ(FILE) (Y(IY), IY = 1, NY)
READ(FILE) ((KSQR(IX,IY), IY = 1, NY), IX = 1, NX)
READ(FILE) ((PHI(IX,IY), IY = 1, NY), IX = 1, NX)

RETURN

END

SUBROUTINE PRVPHI

Reads PHI(NX,NY) unformatted from logical unit FILE. Return code RC set to 1 if size mismatch detected.

Parameters: As above

SUBROUTINE PRVPHI(PHI,NX,NY,FILE,RC)

INTEGER NX, NY
REAL*8 PHI(NX,NY)

INTEGER IX, IY, FILE, GNX, GNY, RC

RC = 0
READ(FILE) GNX, GNY
IF( GNX .NE. NX .OR. GNY .NE. NY ) GO TO 10
READ(FILE) ((PHI(IX,IY) , IY = 1 , NY) , IX = 1 , NX)

RETURN

10 RC = 1
RETURN

END

SUBROUTINE OUTRS1

Writing KSOR(NX,NY), PHI(NX,NY), X(NX), Y(NY) unformatted
on logical unit FILE.

Parameters: As above

SUBROUTINE OUTRS1(KSOR,PHI,X,Y,NX,NY,FILE)

INTEGER NX, NY
REAL*8 KSOR(NX,NY), PHI(NX,NY), X(NX), Y(NY)

INTEGER IX, IXFIN, IXST, IY, IYFIN, IYST, FILE,
* OUTNX, OUTNY

IXST = 2
IXFIN = NX
OUTNX = IXFIN - IXST + 1
IYST = 2
IYFIN = NY - 1
OUTNY = IYFIN - IYST + 1
WRITE(FILE) OUTNX, OUTNY
WRITE(FILE) (X(IX) , IX = IXST , IXFIN)
WRITE(FILE) (Y(IY) , IY = IYST , IYFIN)
WRITE(FILE) ((KSOR(IX,IY) , IY = IYST , IYFIN)
* IX = IXST , IXFIN)
WRITE(FILE) ((PHI(IX,IY) , IY = IYST , IYFIN)
* IX = IXST , IXFIN)
SUBROUTINE OUTRS2

C Writes PHI(NX,NY) unformatted on logical unit FILE.
C Parameters: As above

SUBROUTINE OUTRS2(PHI,NX,NY,FILE)

INTEGER NX, NY
REAL*8 PHI(NX,NY)
INTEGER IX, IY, FILE

WRITE(FILE) NX, NY
WRITE(FILE) ((PHI(IX,IY) , IY = 1 , NY) , IX = 1 , NX)
RETURN
END
MISCELLANEOUS ROUTINES

FUNCTION BIG

Returns maximum of X and Y.

DOUBLE PRECISION FUNCTION BIG(X,Y)

REAL*8 X, Y

BIG = X
IF( Y .GT. X ) BIG = Y
RETURN
END

FUNCTION SMALL

Returns minimum of X and Y.

DOUBLE PRECISION FUNCTION SMALL(X,Y)

REAL*8 X, Y

SMALL = X
IF( Y .LT. X ) SMALL = Y
RETURN
END
FUNCTION CEIL

INTEGER FUNCTION CEIL(X)

REAL*8 X, X1

X1 = X + 0.99999D0
CEIL = X1
RETURN
END

FUNCTION TRNCAT

DOUBLE PRECISION FUNCTION TRNCAT(X)

REAL*8 DFLOAT, X

INTEGER IX

IX = X
TRNCAT = DFLOAT(IX)
RETURN
END

Returns smallest integer .GE. X

Returns largest integral double precision number .LE. X

Routines called:
DFLOAT